

# Theory of Symbolic Expressions, II

By

Masahiko SATO\*

## Abstract

A new domain  $S$  of symbolic expressions is introduced and its structure is studied formally. To study  $S$  formally an intuitionistic first order theory,  $SA$ , is introduced.  $SA$  is a theory adequate for developing elementary metamathematics within it. Gödel's second incompleteness theorem is proved formally within  $SA$  to show the adequacy. A modified version of Post-Smullyan's formal system is used to define basic concepts in  $SA$ . The close relation between formal systems and the logic programming language Qute is also pointed out.

## Introduction

This paper is a continuation to our former paper Sato [10], which we will refer to as I in the sequel. In this paper we continue our study of the domain  $S$  of symbolic expressions. In I we studied the domain  $S$  informally, but in this paper we treat  $S$  formally within a formal mathematical theory.

Through our attempts at formalization of the domain  $S$  we encountered several technical difficulties. Most of these difficulties came from the fact that cons of 0 and 0 was again 0. (We will not go into the details of the difficulties, but we just mention that they are mostly related to the induction schema on sexps.) We were therefore forced to reconsider the domain itself, and by a simple modification (or, rather simplification) on the definition of symbolic expressions we got a new domain. This domain, which we will denote by the symbol  $S$ , will be the objective of our study in this paper. We will refer to our old domain of symbolic expressions which we studied in I as  $S_{old}$ .

This paper can be read without any familiarity with I. We should, however, remark that these two domains are very similar to each other and we will

---

This paper is based on the result of activities of working groups for the Fifth Generation Computer Systems Projects.

Communicated by S. Takasu, April 25, 1984.

\* Department of Information Science, Faculty of Science, University of Tokyo, Tokyo, 113, Japan.

study our new domain with the same spirit as in I.

Besides our previous works [10, 11, 12], the domain of symbolic expressions recently attracted the attention of some logicians. Feferman [4] introduced second order theories of symbolic expressions and showed that elementary metamathematics can be naturally developed within his systems. Hayashi [7] also introduced a theory of symbolic expressions and gave sound foundation for his computer implemented system that synthesizes a LISP program from the constructive proof of its specification. The most important reason for the choice of symbolic expressions as the domain of discourse is because they provide a natural and easy way of encoding the metamathematical entities such as proofs or programs. We will adopt the domain of symbolic expressions as our basic objects of our study for the very same reason.

The paper is organized as follows. In Section 1, we introduce our new domain  $\mathbf{S}$  of symbolic expressions informally. In Section 2, we introduce the concept of a *formal system*, which is a simplified version of the corresponding concept we studied in I. As in I, formal systems will play fundamental roles in our formal study of  $\mathbf{S}$ . We also point out that a formal system is essentially equivalent to a program written in a logic programming language. In Section 3 we introduce a formal theory of symbolic expressions which we call  $\mathbf{BSA}$  (for *Basic Symbolic Arithmetic*). We also explain the intended interpretation of the theory. In Section 4 we introduce a formal system  $\mathbf{FOT}$  (for *First Order Theory*). In  $\mathbf{FOT}$  we can regard any *sexp* as an *axiom system* and we can define arbitrary axiom system with countable first-order language over intuitionistic (or classical) logic by taking suitable *sexp* for the axiom system.

Formal development of mathematics and metamathematics on the domain  $\mathbf{S}$  begins from Section 5. In Section 5 we introduce an axiom system  $\mathbf{SA}$  (for *Symbolic Arithmetic*), which is a conservative extension of  $\mathbf{BSA}$ , as a system which is adequate for the formal development of metamathematics within the system.

In Section 6 we develop simple mathematics within  $\mathbf{SA}$  as a preparation for Section 7 where we develop elementary metamathematics within  $\mathbf{SA}$ . Section 8 will be devoted to the formal proof of some of Gödel's incompleteness theorems.

## §1. Symbolic Expressions

## 1.1. sexps

We define symbolic expressions (*sexps*, for short) by the following inductive clauses:

1.  $*$  is a *sexp*.
2. If  $s$  and  $t$  are *sexps* then  $\text{cons}(s, t)$  is a *sexp*.
3. If  $s$  and  $t$  are *sexps* then  $\text{snoc}(s, t)$  is a *sexp*.

All the *sexps* are constructed by finitely many applications of the above three clauses, and *sexps* constructed differently are distinct. We denote the set of all the *sexps* by  $\mathbb{S}$ . We denote the image of the function  $\text{cons}$  by  $\mathbb{M}$  and that of  $\text{snoc}$  by  $\mathbb{A}$ . We then have two bijective functions:

$$\text{cons}: \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{M}$$

$$\text{snoc}: \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{A}$$

Moreover, by the construction of  $\mathbb{S}$ , we see that  $\mathbb{S}$  is the union of three mutually disjoint sets  $\{*\}$ ,  $\mathbb{M}$  and  $\mathbb{A}$ . In other words,  $\mathbb{S}$  satisfies the following domain equation:

$$\mathbb{S} \equiv \{*\} + \mathbb{A} + \mathbb{M} \cong \{*\} + \mathbb{S} \times \mathbb{S} + \mathbb{S} \times \mathbb{S}$$

We will use the symbol ' $\equiv$ ' as informal equality symbol, and will reserve the symbol '=' for the formal equality sign. Elements in  $\mathbb{M}$  are called *molecules* and those in  $\mathbb{A}$  are called *atoms* and  $*$  is called *nil*. We define two total functions,  $\text{car}$  and  $\text{cdr}$ , on  $\mathbb{M}$  by the equations:

$$\text{car}(\text{cons}(s, t)) \equiv s$$

$$\text{cdr}(\text{cons}(s, t)) \equiv t$$

Similarly we define two total functions,  $\text{cbr}$  and  $\text{ccr}$ , on  $\mathbb{A}$  by the equations:

$$\text{cbr}(\text{snoc}(s, t)) \equiv s$$

$$\text{ccr}(\text{snoc}(s, t)) \equiv t$$

Compositions of the functions  $\text{car}$ ,  $\text{cbr}$ ,  $\text{ccr}$  and  $\text{cdr}$  will be abbreviated following the convention in LISP. For instance:

$$\text{cabcdr}(t) \equiv \text{car}(\text{cbr}(\text{ccr}(\text{cdr}(t))))$$

We must introduce some notations for *sexp*. The so-called dot notation and list notation introduced below is fundamental.

$$[. t] \equiv t$$

$$\begin{aligned} [t_1, \dots, t_n \cdot t_{n+1}] &\equiv \text{cons}(t_1, [t_2, \dots, t_n \cdot t_{n+1}]) \quad (n \geq 1) \\ [t_1, \dots, t_n] &\equiv [t_1, \dots, t_n \cdot *] \quad (n \geq 0) \end{aligned}$$

In particular we have

$$\begin{aligned} [s \cdot t] &\equiv \text{cons}(s, t) \\ [ ] &\equiv * \end{aligned}$$

A sexp of the form  $[t_1, \dots, t_n]$  will be called a *list*. We will also use the following abbreviations.

$$\begin{aligned} s[t] &\text{ for } [s \cdot t] \\ s[t_1, \dots, t_n \cdot t_{n+1}] &\text{ for } [s, t_1, \dots, t_n \cdot t_{n+1}] \\ s[t_1, \dots, t_n] &\text{ for } [s, t_1, \dots, t_n] \end{aligned}$$

For *snoc*, we only use the following notation

$$(s \cdot t) \equiv \text{snoc}(s, t)$$

Parentheses will also be used for grouping. Thus  $(t)$  will *not* denote  $\text{snoc}(t, *)$  but will denote  $t$ . (Readers of our previous papers should note the change of notations.)

The basic *induction schema* on  $\mathbb{S}$  can be stated as follows. Let  $\Phi(t)$  be a proposition about a sexp  $t$ . Then we may conclude that  $\Phi(t)$  holds for any  $t$ , if we can prove the following three propositions.

- (i)  $\Phi(*)$
- (ii) If  $\Phi(s)$  and  $\Phi(t)$  then  $\Phi([s \cdot t])$
- (iii) If  $\Phi(s)$  and  $\Phi(t)$  then  $\Phi((s \cdot t))$

## 1.2. symbols and variables

An atom of the form

$$(* \cdot x)$$

will be called a *symbol*. Let  $\Sigma$  be the set of 128 ASCII characters. We define an injective function  $\rho: \Sigma \rightarrow \mathbb{M}$  by using 7 bit ASCII codes, regarding  $*$  as 0 and  $[*]$  as 1. For instance, we have

$$\begin{aligned} \rho(a) &\equiv [[*], [*], *, *, *, *, [*]] \\ \rho(1) &\equiv [*], [*], [*], *, *, *, [*] \end{aligned}$$

We extend  $\rho$  homomorphically to  $\Sigma^*$ , i.e., we define  $\rho^*: \Sigma^* \rightarrow \mathbb{M}$  by  $\rho^*(\sigma_1 \cdots \sigma_k) \equiv [\rho(\sigma_1), \dots, \rho(\sigma_k)]$  ( $\sigma_i \in \Sigma$ ). Now consider a string of alphanumeric characters such that

- (i) its length is longer than 1,
- (ii) it begins with a lowercase character and
- (iii) its second character is a non-numeric character.

Such a string will be called a *name*. Let  $\pi$  be a name. Then, by definition,  $\pi$  denotes the symbol

$$(*.[*.\rho^*(\pi)])$$

An atom of the form

$$(\text{var}.x)$$

is called a *variable*. (Note that ‘var’ denotes a specific symbol. See Example 1.1 below.) We introduce notations for variables. A string of alphanumeric characters such that

- (i) it begins with an uppercase character, or
- (ii) it consists of a single lowercase character, or
- (iii) its first character is lowercase and its second character is a numeral

denotes a variable as follows. Let  $\pi$  be such a string. Then, by definition,  $\pi$  denotes the variable

$$(\text{var}.(*.[*.\rho^*(\pi)])$$

We will regard the under score character ‘\_’ as a lower case character for convenience.

**Example 1.1.**

$$\text{var} \equiv (*.[*.[[[[*], [*], [*], *, [*], [*], *], [[[*], [*], *, *, *, *, [*]],$$

$$[[[*], [*], [*], *, *, [*], *]])]$$

$$\text{Var} \equiv (\text{var}.(*.[*.[[[[*], *, [*], *, [*], [*], *], [[[*], [*], *, *, *, *, [*]],$$

$$[[[*], [*], [*], *, *, [*], *]])]) \quad \square$$

## §2. Formal Systems

### 2.1. formal system

In I, we have defined the concept of a *formal system*. Here we will redefine a formal system by giving a simpler definition of it. As explained in I, our concept of a formal system has its origin in Smullyan [13]. However, unlike Smullyan’s, our formal system will be defined directly as a *sexp*. This has the advantage of making the definition of a universal formal system simpler. Another practically very important aspect of our concept of a formal system is that

it can be quite naturally viewed as a so-called logic program. This means that we can execute formal systems on a computer if we have an interpreter for them. In fact, Takafumi Sakurai of the University of Tokyo implemented such an interpreter. (See [12].) We can therefore use formal systems both as theoretically and practically basic tools for our study of symbolic expressions.

*Note.* When we introduced formal systems in I, we were ignorant of the programming language PROLOG. But after we had submitted I for publication, we knew the existence of the language. Since it was clear, for any one who knows both PROLOG and Post-Smullyan's formal system (or, the concept of inductive definition), that they are essentially the same, we asked T. Sakurai to implement an interpreter for our formal systems which we introduced in I. The interpreter was named Hyperprolog, and it was used to debug the definition of Ref which we gave in I. In this way we could correct bugs in our formal systems in the stage of proof reading. We believe that the existence of an interpreter is essential for finding and correcting such *bugs*. We also remark that Hyperprolog was quite useful in designing our new formal system, which we are about to explain, since it can be simulated on Hyperprolog. Finally we remark that we have designed a new programming language called Qute which can compute relations defined by our new formal system. Qute was also implemented by T. Sakurai. (See Sato and Sakurai [12].)  $\square$

Now let us define our formal system. We will call, by definition, any sexp a *formal system*. Our objective, then, is to define a relation *proves*( $p, a, FS$ ) which holds among certain triples  $p, a, FS$  of sexps where the sexp  $FS$  is treated as a formal system. We will employ informal inductive definitions to define the relation *proves*. We will say that  $p$  is a *proof* of  $a$  in the *formal system*  $FS$ , if *proves*( $p, a, FS$ ) holds. We write:

$$p \vdash_{FS} a \text{ for } \textit{proves} (p, a, FS)$$

We will say that  $a$  is a *theorem* in  $FS$  if *proves* ( $p, a, FS$ ) holds for some  $p$ , and will use the notation:

$$\vdash_{FS} a$$

for it.

## 2.2 inductive definitions

As an example of informal inductive definition, let us define the relation *member* ( $x, L$ ) which means that  $x$  is a member of  $L$ :

- (M1)  $\implies member(x, [x.L])$   
 (M2)  $member(x, L) \implies member(x, [y.L])$

(M1) means that the relation  $member(x, [x.L])$  holds unconditionally for any sexp  $x$  and  $L$ , and (M2) says that if the relation  $member(x, L)$  holds then the relation  $member(x, [y.L])$  also holds for any sexp  $x, L$  and  $y$ . We have omitted the usual extremal clause which states that the relation  $member(x, L)$  holds only when it can be shown to be so by finitely many applications of the clauses (M1) and (M2).

Let us now consider about the nature of (informal) inductive definitions in general. All inductive definitions which we consider in this paper consist of a finite set of clauses (or, rules) of the form

$$(\Gamma) \quad \gamma_1, \dots, \gamma_n \implies \gamma$$

where  $n \geq 0$  and  $\Gamma$  is the name of the clause which is used to identify the clause. For example, in (M1)  $n$  is 0 and in (M2)  $n$  is 1. Suppose we have a finite set of inductive clauses like above, and we could conclude that a certain specific relation holds among specific sexps from these inductive clauses. Let us write our conclusion as  $\alpha$ . (If our set of inductive clauses consist only of (M1) and (M2) above, then  $\alpha$  is of the form  $member(x, L)$  where  $x$  and  $L$  are certain specific sexps such as orange or [apple, orange].) We now show that we can associate with  $\alpha$  an informal proof  $\Pi$  of  $\alpha$ . According to the extremal clause,  $\alpha$  is obtained by applying our inductive clauses finitely many times. Let  $(\Gamma)$  be the last applied clause. Since the clause  $(\Gamma)$  is schematic, when we apply  $(\Gamma)$  we must also specify for each schematic variables  $x_i$  a sexp  $v_i$  as its value. Let  $x_1, \dots, x_k$  be an enumeration of schematic variables occurring in  $(\Gamma)$  and let

$$\Delta \equiv \langle x_1 := v_1, \dots, x_k := v_k \rangle$$

By substituting  $v_i$  for  $x_i$ , we can obtain the following instance of  $(\Gamma)$ :

$$(\Gamma_\Delta) \quad \alpha_1, \dots, \alpha_n \implies \alpha$$

Note that the conclusion of  $(\Gamma_\Delta)$  must be  $\alpha$  by our assumption that  $\alpha$  is obtained by applying (an instance of)  $(\Gamma)$ . That  $(\Gamma_\Delta)$  is applicable also means that each  $\alpha_i$  has already been shown to hold by applying inductive clauses finitely many times. Since the number of applications of inductive clauses which was used to show  $\alpha_i$  is smaller than that was required to show  $\alpha$ , we may assume, as induction hypothesis, that we have an informal proof  $\Pi_i$  of  $\alpha_i$  for each  $1 \leq i \leq n$ . Using

these data, we can construct a proof  $\Pi$  of  $\alpha$  as the figure of the form :

$$\frac{\Pi_1 \cdots \Pi_n}{(\Gamma)\Delta}$$

**Example 2.1.**

From (M1) and (M2), we can conclude that *member*(orange, [apple, orange]) holds, and we have the following proof associated with this.

$$\frac{\overline{(\text{M1})\langle x := \text{orange}, L := [ ] \rangle}}{(\text{M2})\langle x := \text{orange}, y := \text{apple}, L := [\text{orange}] \rangle} \quad \square$$

### 2.3. definition of the relation proves

Based on this intuitive idea of informal *proof*, we define the relation *proves* etc. as follows. First we define *ne* (for *not equal*) which has the property that *ne*( $x, y$ ) holds iff  $x$  and  $y$  are two distinct sexps.

- (N1)  $\implies ne(*, [u . v])$
- (N2)  $\implies ne(*, (u . v))$
- (N3)  $\implies ne([s . t], *)$
- (N4)  $\implies ne((s . t), *)$
- (N5)  $\implies ne([s . t], (u . v))$
- (N6)  $\implies ne((s . t), [u . v])$
- (N7)  $ne(s, u) \implies ne([s . t], [u . v])$
- (N8)  $ne(t, v) \implies ne([s . t], [u . v])$
- (N9)  $ne(s, u) \implies ne((s . t), (u . v))$
- (N10)  $ne(t, v) \implies ne((s . t), (u . v))$

We next define *assoc* which is used to get the value of a variable from a given environment.

- (A1)  $\implies assoc(x, [[x . v] . L], v)$
- (A2)  $ne(x, y), assoc(x, L, v) \implies assoc(x, [[y . w] . L], v)$ .

**Example 2.2.**

*assoc*(c, [[a . apple], [b . banana], [c . carrot]], carrot)  $\square$

The relation *get* is used to extract the  $i$ -th member of a list  $L$ .

- (G1)  $\implies get(*, [v . L], v)$
- (G2)  $get(i, L, v) \implies get([* . i], [w . L], v)$

**Example 2.3.**

*get*([\* , \*], [lisp, prolog, quote], quote)  $\square$



The following relation *eval* gives a simple evaluator of a sexp under a certain environment. Substitution of values to variables can be simulated by *eval*.

- (E1)  $assoc((var . t), Env, v) \implies eval((var . t), Env, v)$
- (E2)  $\implies eval(*, Env, *)$
- (E3)  $eval(s, Env, u), eval(t, Env, v) \implies eval([s . t], Env, [u . v])$
- (E4)  $eval(s, Env, u), eval(t, Env, v) \implies eval((snoc . [s, t]), Env, (u . v))$
- (E5)  $\implies eval((* . t), Env, (* . t))$
- (E6)  $\implies eval((quote . t), Env, t)$

We will use the following abbreviations for atoms whose *cbr* is *snoc* or *quote*.

- (: *s . t*) for (snoc . [s, t])
- (: *t*) for (snoc . [t, \*])
- '*t* for (quote . t)

**Example 2.4.**

$$eval([x, of, y, and, z, is, '(apple . orange)],$$

$$[[x . snoc], [y . apple], [z . orange]],$$

$$[snoc, of, apple, and, orange, is, (apple . orange)]) \quad \square$$

In terms of these relations we can now define *proves* and *lproves*.

- (L1)  $\implies lproves([ ], [ ], FS)$
- (L2)  $proves(p, a, FS), lproves(P, A, FS)$   
 $\implies lproves([p . P], [a . A], FS)$
- (P1)  $assoc(Prd, FS, R), get(i, R, [c . C]), eval(c, Env, a),$   
 $eval(C, Env, A), lproves(P, A, FS)$   
 $\implies proves([[Prd, i, Env] . P], [Prd . a], FS)$

We can also define the relation  $\vdash_{FS} a$  by the following inductive definition.

- (T1)  $proves(p, a, FS) \implies theorem(a, FS)$

We show by an example how our intuitive idea of proof has been formalized. Recall that the relation *assoc* was defined by the two clauses (A1) and (A2) and that its definition depends also on the relation *ne*. Since *ne* has 10 clauses ((N1)–(N10)), we need 12 clauses to define *assoc*. We formalize these 12 clauses in two steps. In the first step we formalize clauses (A1) and (A2) into a sexp *Assoc* and clauses (N1)–(N10) into a sexp *Ne*. In the second step we obtain a formal sytem  $[Assoc, Ne]$  as a formalization of *assoc* and *ne*. The sexp *Assoc*, which is the translation of clauses (A1) and (A2), is defined as follows:

$$\begin{aligned}
 & [ \text{assoc} \\
 & \quad , [[x, [[x.v].L], v]] \\
 & \quad , [[x, [[y.w].L], v] \\
 & \quad \quad , \text{ne}[x, y] \\
 & \quad \quad , \text{assoc}[x, L, v]] \\
 & ]
 \end{aligned}$$

We explain the general mechanism of our translation of clauses. We translate clauses that are used to define a same relation into a single sexp. We therefore translate (A1) and (A2) into *Assoc* and (N1)–(N10) into *Ne*. Recall that each clause is of the form:

$$\gamma_1, \dots, \gamma_n \implies \gamma$$

and that the general form of  $\gamma$  or  $\gamma_i$  is:

$$\text{Prd}(\text{Arg}_1, \dots, \text{Arg}_k)$$

We translate *Prd* into corresponding symbol. For instance *assoc* is translated into ‘assoc’. *Arg*’s are translated as follows. Since *Arg* is a schematic expression for sexp it has one of the following forms: (i) a schematic variable, (ii) \*, (iii)  $[\alpha.\beta]$ , (iv)  $(\alpha.\beta)$ . In case of (i) we translate it into corresponding (formal) variable. Thus  $x$  is translated into ‘x’. If *Arg* is \* then it is translation into \*. If *Arg* is of the form (iii), its translation is  $[\alpha^*.\beta^*]$  where  $\alpha^*$  ( $\beta^*$ ) is the translation of  $\alpha$  ( $\beta$ , resp.). Similarly, but slightly differently, case (iv) is translated into  $(:\alpha^*.\beta^*)$  if  $\alpha$  in not \* and it is translated into itself if  $\alpha$  is \*. (Since the translation must be one to one, we cannot translate  $(\alpha.\beta)$  into  $(\alpha^*.\beta^*)$  because, then, (ii)–(iv) will leave no room for the translation of schematic variables.) By extending this translation naturally we obtain the above translation of (A1) and (A2). For the sake of readability we introduce the following abbreviation for the above sexp *Assoc*.

$$\begin{aligned}
 & + \text{assoc} \\
 & \quad | x, [[x.v].L], v \\
 & \quad | x, [[y.w].L], v \\
 & \quad \quad - \text{ne}[x, y] \\
 & \quad \quad - \text{assoc}[x, L, v]
 \end{aligned}$$

**Example 2.5.** By the similar idea as above we can translate the informal proof in Example 2.1 into the following formal proof  $p$ :

$$[[\text{member}, [*], [[x . \text{orange}], [y . \text{apple}], [L . [\text{orange}]]]],$$

$$[[\text{member}, *, [[x . \text{orange}], [L . [ ]]]] ] ]$$

Let *Member* be the following sexp:

$$+ \text{ member}$$

$$| x, [x . L]$$

$$| x, [y . L]$$

$$- \text{ member}[x, L]$$

$$;$$

Then we can easily verify that

$$p \vdash_{[\text{Member}]} \text{member}[\text{orange}, [\text{apple}, \text{orange}]]$$

holds and hence

$$\vdash_{[\text{Member}]} \text{member}[\text{orange}, [\text{apple}, \text{orange}]]$$

holds.  $\square$

#### 2.4. universal formal system

By translating the relations we have defined so far we obtain a formal system  $\text{Univ}$  which is universal among all the formal systems. We thus define  $\text{Univ}$  as the sexp:

$$\text{Univ} \equiv [\text{Ne}, \text{Assoc}, \text{Get}, \text{Eval}, \text{Lproves}, \text{Proves}, \text{Theorem}]$$

where *Ne*, *Assoc*, *Get*, *Eval*, *Lproves*, *Proves* and *Theorem* are respectively:

$$+ \text{ ne}$$

$$| *, [u . v]$$

$$| *, (: u . v)$$

$$| [s . t], *$$

$$| (: s . t), *$$

$$| [s . t], (: u . v)$$

$$| (: s . t), [u . v]$$

$$| [s . t], [u . v]$$

$$- \text{ ne}[s, u]$$

$$| [s . t], [u . v]$$

$$- \text{ ne}[t, v]$$

$$| (: s . t), (: u . v)$$

$$- \text{ ne}[s, u]$$

```

    | (: s . t), (: u . v)
      - ne[t, v]
    ;
+ assoc
    | x, [[x . v] . L], v
    | x, [[y . w] . L], v
      - ne[x, y]
      - assoc[x, L, v]
    ;
+ get
    | *, [v . L], v
    | [* . i], [w . L], v
      - get[i, L, v]
    ;
+ eval
    | (: var . t), Env, v
      - assoc[(: var . t), Env, v]
    | *, Env, *
    | [s . t], Env, [u . v]
      - eval[s, Env, u]
      - eval[t, Env, v]
    | (: snoc . [s, t]), Env, (: u . v)
      - eval[s, Env, u]
      - eval[t, Env, v]
    | (: * . t), Env, (: * . t)
    | (: quote . t), Env, t
    ;
+ lproves
    | [ ], [ ], FS
    | [p . P,], [a . A], FS
      - proves[p, a, FS]
      - lproves[P, A, FS]
    ;
+ proves
    | [[Prd, i, Env] . P], [Prd . a], FS
      - assoc[Prd, FS, R]

```

$$\begin{array}{l}
- \text{ get}[i, R, [c \cdot C]] \\
- \text{ eval}[c, \text{Env}, a] \\
- \text{ eval}[C, \text{Env}, A] \\
- \text{ lproves}[P, A, FS] \\
; \\
+ \text{ theorem} \\
\quad | a, FS \\
\quad - \text{ proves}[p, a, FS] \\
;
\end{array}$$

The following theorem establishes that  $\mathbb{U}_{\text{univ}}$  is in fact a universal formal system.

**Theorem 2.1.**

- (i)  $ne(x, y) \iff \vdash_{\mathbb{U}_{\text{univ}}} ne[x, y]$
- (ii)  $assoc(x, L, v) \iff \vdash_{\mathbb{U}_{\text{univ}}} assoc[x, L, v]$
- (iii)  $get(i, L, v) \iff \vdash_{\mathbb{U}_{\text{univ}}} get[i, L, v]$
- (iv)  $eval(t, E, v) \iff \vdash_{\mathbb{U}_{\text{univ}}} eval[t, E, v]$
- (v)  $lproves(P, A, FS) \iff \vdash_{\mathbb{U}_{\text{univ}}} lproves[P, A, FS]$
- (vi)  $proves(p, a, FS) \iff \vdash_{\mathbb{U}_{\text{univ}}} proves[p, a, FS]$
- (vii)  $theorem(a, FS) \iff \vdash_{\mathbb{U}_{\text{univ}}} theorem[a, FS]$

We omit the simple but tedious combinational proof of this theorem. The following corollary is simply a restatement of the last two sentences of this theorem.

**Corollary 2.2.**

- (i)  $p \vdash_{FS} a \iff \vdash_{\mathbb{U}_{\text{univ}}} proves [p, a, FS]$
- (ii)  $\vdash_{FS} a \iff \vdash_{\mathbb{U}_{\text{univ}}} theorem [a, FS]$

### §3. Formal Theory of Symbolic Expressions: BSA

In this section we introduce a formal theory of symbolic expressions which we call BSA (for *Basic Symbolic Arithmetic*). The theory is a first order intuitionistic theory which is proof theoretically equivalent to HA (Heyting arithmetic).

Traditionally, metamathematical entities such as *terms*, *wffs* and *proofs* have been considered as concrete figures which can be displayed on a sheet of paper (with some kind of abstraction which is necessary so as to allow finite but arbitrarily large figures). Our standpoint is, however, not like this but to

regard these entities as symbolic expressions. By taking this standpoint we can define SA formally in terms of a formal system. It is also possible to define BSA in this way, but for the convenience of the reader who is perhaps so accustomed to the traditional approach we first define BSA in the usual way and will then explain how BSA so defined can be isomorphically translated into S. We reserve BSA as the name for the system which we will define as a formal system in Section 3.7, and use  $\mathcal{BSA}$  to denote the theory which we now define by a traditional method.

### 3.1. language

The language of  $\mathcal{BSA}$  consists of the following symbols.

- *individual symbols*: **nil**
- *function symbols*: **cons**, **snoc**
- *pure variables*:  $\text{var}_t$  for each sexp  $t$
- *predicate symbols*: **eq** (equal), **lt** (less than)
- ⊗ *logical symbols*: **and**, **or**, **imply**, **all**, **exist**
- *other symbols*: ( , ), ' , ' (comma), free

### 3.2. variables, terms and wffs

Using the language introduced above, we define syntactic entities of  $\mathcal{BSA}$ . We first define *variables* as follows.

1. For each sexp  $t$ , the pure variable  $\text{var}_t$  is a variable.
2. If  $x$  is a variable then  $\text{free}(x)$  is a variable.

For a variable  $x$  we define its *pure part* as follows.

1. If  $x$  is a pure variable then its pure part is  $x$  itself.
2. If the pure part of  $x$  is  $y$  then the pure part of  $\text{free}(x)$  is also  $y$ .

The definition of *terms* is as follows.

1. A variable is a term.
2. **nil** is a term.
- 3–4. If  $s$  and  $t$  are terms then **cons**( $s$ ,  $t$ ) and **snoc**( $s$ ,  $t$ ) are terms.

We define *wffs* (well formed formulas) as follows.

- 1–2. If  $s$  and  $t$  are terms then **eq**( $s$ ,  $t$ ) and **lt**( $s$ ,  $t$ ) are wffs.
- 3–4. If  $a_1, \dots, a_n$  ( $n \geq 0$ ) are wffs then **and**( $a_1, \dots, a_n$ ) and **or**( $a_1, \dots, a_n$ ) are wffs.
5. If  $a_1, \dots, a_n$  ( $n \geq 0$ ) and  $b$  are wffs then **imply**(( $a_1, \dots, a_n$ ),  $b$ ) is a wff.

6-7. If  $x_1, \dots, x_n$  ( $n \geq 0$ ) are distinct pure variables and  $a$  is a wff then  $\text{all}((x_1, \dots, x_n), a)$  and  $\text{exist}((x_1, \dots, x_n), a)$  are wffs.

A wff is called an *atomic wff* if it is constructed by the clauses 1-2 above, and a wff is called a *quantifier free wff* if it is constructed by the clauses 1-5 above. We will call both a term and a wff as a *designator*.

We will use the following symbols with or without subscripts as syntactic variables for specific syntactic objects.

$x, y, z$  for variables  
 $r, s, t, u, v$  for terms  
 $a, b, c$  for wffs  
 $d, e$  for designators

### 3.3. abbreviations

We introduce the following abbreviations.

#  $x$  for  $\text{free}(x)$   
 $s = t$  for  $\text{eq}(s, t)$   
 $s < t$  for  $\text{lt}(s, t)$   
 $s \leq t$  for  $\text{or}(\text{lt}(s, t), \text{eq}(s, t))$   
 $a_1 \wedge \dots \wedge a_n$  for  $\text{and}(a_1, \dots, a_n)$   
 $a_1 \vee \dots \vee a_n$  for  $\text{or}(a_1, \dots, a_n)$   
 $a_1, \dots, a_n \rightarrow b$  for  $\text{imply}((a_1, \dots, a_n), b)$   
 $a \leftrightarrow b$  for  $\text{and}(\text{imply}((a), b), \text{imply}((b), a))$   
 $\forall(x_1, \dots, x_n; a)$  for  $\text{all}((x_1, \dots, x_n), a)$   
 $\exists(x_1, \dots, x_n; a)$  for  $\text{exist}((x_1, \dots, x_n), a)$

We assume that the binding power of the operators  $\wedge$ ,  $\vee$  and  $\rightarrow$  decrease in this order, and we insert parentheses when necessary to insure unambiguous reading.

### 3.4. substitutions and free variables

Let  $t$  be a term,  $x$  be a variable and  $d$  be a designator. We then define a designator  $e$  which we call the result of *substituting  $t$  for  $x$  in  $d$*  as follows. The definition requires one auxiliary concept, namely, the *elevation of a term with respect to a finite sequence of pure variables*, which we also define below.

- I.1.1. If  $d$  is  $x$  then  $e$  is  $t$ .  
 I.1.2. If  $d$  is a variable other than  $x$  then  $e$  is  $d$ .

- I.2. If  $d$  is **nil** then  $e$  is **nil**.
- I.3. If  $d$  is **cons**( $t_1, t_2$ ) and  $e_1$  ( $e_2$ ) is the results of substituting  $t$  for  $x$  in  $t_1$  ( $t_2$ , resp.) then  $e$  is **cons**( $e_1, e_2$ ).
- I.4. If  $d$  is **snoc**( $t_1, t_2$ ) and  $e_1$  ( $e_2$ ) is the result of substituting  $t$  for  $x$  in  $t_1$  ( $t_2$ , resp.) then  $e$  is **snoc**( $e_1, e_2$ ).
- II.1. If  $d$  is **eq**( $t_1, t_2$ ) and  $e_1$  ( $e_2$ ) is the result of substituting  $t$  for  $x$  in  $t_1$  ( $t_2$ , resp.) then  $e$  is **eq**( $e_1, e_2$ ).
- II.2. If  $d$  is **lt**( $t_1, t_2$ ) and  $e_1$  ( $e_2$ ) is the result of substituting  $t$  for  $x$  in  $t_1$  ( $t_2$ , resp.) then  $e$  is **lt**( $e_1, e_2$ ).
- II.3–4. If  $d$  is **and**( $a_1, \dots, a_n$ ) or **or**( $a_1, \dots, a_n$ ) and  $e_i$  ( $1 \leq i \leq n$ ) is the result of substituting  $t$  for  $x$  in  $a_i$  then  $e$  is **and**( $e_1, \dots, e_n$ ) (**or**( $e_1, \dots, e_n$ ), resp.).
- II.5. If  $d$  is **imply**(( $a_1, \dots, a_n$ ),  $b$ ),  $e_i$  ( $1 \leq i \leq n$ ) is the result of substituting  $t$  for  $x$  in  $a_i$  and  $c$  is the result of substituting  $t$  for  $x$  in  $b$  then  $e$  is **imply**(( $e_1, \dots, e_n$ ),  $c$ ).
- II.6. If  $d$  is **all**(( $x_1, \dots, x_n$ ),  $a$ ),  $u$  ( $y$ ) is the elevation of  $t$  ( $x$ , resp.) with respect to the sequence of pure variables  $x_1, \dots, x_n$  and  $b$  is the result of substituting  $u$  for  $y$  in  $a$  then  $e$  is **all**(( $x_1, \dots, x_n$ ),  $b$ ).
- II.7. If  $d$  is **exist**(( $x_1, \dots, x_n$ ),  $a$ ),  $u$  ( $y$ ) is the elevation of  $t$  ( $x$ , resp.) with respect to the sequence of pure variables  $x_1, \dots, x_n$  and  $b$  is the result of substituting  $u$  for  $y$  in  $a$  then  $e$  is **exist**(( $x_1, \dots, x_n$ ),  $b$ ).

Let  $t$  be a term and  $x_1, \dots, x_n$  ( $n \geq 0$ ) be a sequence of distinct pure variables. We define a term  $u$  which we call the elevation of  $t$  with respect to  $x_1, \dots, x_n$  as follows.

- 1.1. If  $t$  is a variable whose pure part is  $x_i$  for some  $i$  ( $1 \leq i \leq n$ ) then  $u$  is **free**( $t$ ).
- 1.2. If  $t$  is a variable whose pure part does not appear in the sequence  $x_1, \dots, x_n$  then  $u$  is  $t$ .
- 2. If  $t$  is **nil** then  $u$  is **nil**.
- 3. If  $t$  is a term **cons**( $t_1, t_2$ ) and  $u_1$  ( $u_2$ ) is the elevation of  $t_1$  ( $t_2$ , resp.) with respect to the sequence  $x_1, \dots, x_n$  then  $u$  is **cons**( $u_1, u_2$ ).
- 4. If  $t$  is a term **snoc**( $t_1, t_2$ ) and  $u_1$  ( $u_2$ ) is the elevation of  $t_1$  ( $t_2$ , resp.) with respect to the sequence  $x_1, \dots, x_n$  then  $u$  is **snoc**( $u_1, u_2$ ).



That the result of substituting a term for a variable in a designator is again a designator of the same type can be proved easily by induction. (To prove this, one must also prove that the elevation of a term with respect to a sequence of distinct pure variables is also a term.)

**Example 3.1.**

(i) Let  $x$  and  $y$  be distinct pure variables and let  $a$  be the wff  $\exists(x; x = y)$ . Let us substitute  $x$  for  $y$  in  $a$ . To do so, we must first compute the elevations of  $x$  and  $y$  with respect to  $x$ . They are  $\#x$  and  $y$  respectively. Now the result of substituting  $\#x$  for  $y$  in  $x = y$  is  $x = \#x$ . Thus we have that  $\exists(x; x = \#x)$  is the result of substituting  $x$  for  $y$  in  $a$ . Let us call this wff  $b$ . Then the reader should verify that the result of substituting  $y$  for  $x$  in  $b$  is  $a$ .

(ii) Let  $z$  be a variable distinct from  $x$  and  $y$  above and consider the wff  $\exists(x, y; z = \text{coms}(x, y))$ . Then the result of substituting the term  $\text{coms}(x, y)$  for  $z$  in this wff is calculated similarly as above and we obtain the wff  $\exists(x, y; \text{coms}(\#x, \#y) = \text{coms}(x, y))$ .  $\square$

*Remark.* As can be seen in the above examples we have avoided the problem of the collision of variables by introducing a systematic way of referring to a *non-local* variable that happens to have the same name as one of the *local* variables. We remark that our method is a generalization of the method due to de Bruijn [3].  $\square$

We can define simultaneous substitution similarly. Let  $t_1, \dots, t_n$  be a sequence of terms,  $x_1, \dots, x_n$  be a sequence of distinct variables and let  $d$  be a designator. We will use the notation  $d_{x_1, \dots, x_n}[t_1, \dots, t_n]$  to denote the result of simultaneously substituting  $t_1, \dots, t_n$  for  $x_1, \dots, x_n$  in  $d$ .

We say that a variable  $x$  *occurs free* in a designator  $d$  if  $d_x[\text{nil}]$  is distinct from  $d$ . A designator is said to be *closed* if no variables occur free in it.

We need the following concept in the definition of proofs below. Let  $t$  be a term,  $x$  be a variable and  $d$  be a designator. We then define a designator  $e$  which we call the result of *bind substituting*  $t$  for  $x$  in  $d$  as follows. The definition goes completely in parallel with the definition of substitution except for the clause I.1.2. We therefore only give the clause I.1.2 below.

I.1.2. If  $d$  is a variable other than  $x$  then:

if the pure parts of  $d$  and  $x$  are the same then:

if  $d$  is a pure variable then  $e$  is  $d$ ;

if  $x$  is a pure variable then  $e$  is defined so that  $d \equiv \#e$ ;

if  $x \equiv \#x_1$  and  $d \equiv \#d_1$  then  $e$  is  $\#e_1$  where  $e_1$  is the result of bind substituting  $t$  for  $x_1$  in  $d_1$ ;

if the pure parts of  $d$  and  $x$  are distinct then  $e$  is  $d$ .

Let  $t_1, \dots, t_n$  be a sequence of terms,  $x_1, \dots, x_n$  be a sequence of variables whose pure parts are distinct and  $d$  be a designator. We can define the result of simultaneously bind substituting  $t_1, \dots, t_n$  for  $x_1, \dots, x_n$  in  $d$  similarly as above, and we use the notation  $d_{x_1, \dots, x_n} \llbracket t_1, \dots, t_n \rrbracket$  for it.

### 3.5. proofs

We formulate our fromal theory *BSA* in natural deduction style. Since we eventually give a precise definition of *BSA* using a formal system, we give here an informal definition in terms of schematic inference rules. Namely an inference rule is a figure of the form:

$$\frac{a_1 \dots a_n}{a} \quad n \geq 0$$

where  $a_i, a$  are formulas.  $a_i$  may have *assumptions* that are *discharged* at this inference rule, and we show such assumptions by enclosing them by brackets. We call  $a_1, \dots, a_n$  the *premises* and  $a$  the *consequence* of the inference rule. We first collect logical rules. The logic we use is the first order intuitionistic logic with equality.

$$\begin{array}{ll} (\wedge I) \frac{a_1 \dots a_n}{a_1 \wedge \dots \wedge a_n} & (\wedge E)_i \frac{a_1 \wedge \dots \wedge a_n}{a_i} \quad 1 \leq i \leq n \\ & [a_1] \dots [a_n] \\ (\vee I)_i \frac{a_i}{a_1 \vee \dots \vee a_n} \quad 1 \leq i \leq n & (\vee E) \frac{a_1 \vee \dots \vee a_n \quad c}{c} \\ & [a_1] \dots [a_n] \\ (\rightarrow I) \frac{b}{a_1, \dots, a_n \rightarrow b} & (\rightarrow E) \frac{a_1, \dots, a_n \rightarrow b \quad a_1 \dots a_n}{b} \\ (\forall I) \frac{a_{x_1, \dots, x_n} \llbracket y_1, \dots, y_n \rrbracket}{\forall (x_1, \dots, x_n; a)} & (\forall E) \frac{\forall (x_1, \dots, x_n; a)}{a_{x_1, \dots, x_n} \llbracket t_1, \dots, t_n \rrbracket} \\ & [a_{x_1, \dots, x_n} \llbracket y_1, \dots, y_n \rrbracket] \\ (\exists I) \frac{a_{x_1, \dots, x_n} \llbracket t_1, \dots, t_n \rrbracket}{\exists (x_1, \dots, x_n; a)} & (\exists E) \frac{\exists (x_1, \dots, x_n; a) \quad b}{b} \\ (=) \frac{}{t = t} & (= subst) \frac{a_{x_1, \dots, x_n} [s_1, \dots, s_n] \quad s_1 = t_1 \dots s_n = t_n}{a_{x_1, \dots, x_n} [t_1, \dots, t_n]} \end{array}$$

In the above rules the variables  $x_1, \dots, x_n$  must be distinct pure variables. The

variables  $y_1, \dots, y_n$  must be distinct and must satisfy the *eigen variables conditions*. That is, in  $(\forall I)$ , they must not occur free in  $\forall(x_1, \dots, x_n; a)$  or in any assumption on which  $a_{x_1, \dots, x_n} [y_1, \dots, y_n]$  depends, and in  $(\exists E)$ , they must not occur free in  $\exists(x_1, \dots, x_n; a)$ ,  $b$  or any assumption other than  $a_{x_1, \dots, x_n} [y_1, \dots, y_n]$  on which the premise  $b$  depends.

Note that we may regard the wffs  $\text{and}()$  and  $\text{or}()$  as representing the truth values *true* and *false* respectively by letting  $n$  to be 0 in  $(\wedge I)$  and  $(\vee E)$ . For this reason, we will use  $\perp$  as an abbreviation for  $\text{or}()$ ,  $a \neg$  for  $a \rightarrow \perp$  and  $s \neq t$  for  $\neg(s = t)$ .

The remaining rules are specific to the theory  $BSA$ . First we consider the rules for equality.

$$\begin{aligned} & (\text{cons} \neq \text{nil}) \frac{\text{cons}(s, t) = \text{nil}}{\perp} \quad (\text{snoc} \neq \text{nil}) \frac{\text{snoc}(s, t) = \text{nil}}{\perp} \\ & (\text{cons} \neq \text{snoc}) \frac{\text{cons}(s, t) = \text{snoc}(u, v)}{\perp} \\ & (\text{cons} = \text{cons})_i \frac{\text{cons}(s_1, s_2) = \text{cons}(t_1, t_2)}{s_i = t_i} \quad i = 1, 2 \\ & (\text{snoc} = \text{snoc})_i \frac{\text{snoc}(s_1, s_2) = \text{snoc}(t_1, t_2)}{s_i = t_i} \quad i = 1, 2 \end{aligned}$$

Next we collect rules for  $<$  (less than).

$$\begin{aligned} & (<*) \frac{r < *}{\perp} \quad (< \text{snoc}) \frac{r < \text{snoc}(s, t)}{\perp} \\ & (<)_i \frac{}{t_i < \text{cons}(t_1, t_2)} \quad i = 1, 2 \quad (< \text{cons})_i \frac{s < t_i}{s < \text{cons}(t_1, t_2)} \quad i = 1, 2 \\ & \quad \quad \quad [r = s] \quad [r < s] \quad [r = t] \quad [r < t] \\ & (< \text{cons } E) \frac{r < \text{cons}(s, t) \quad c \quad c \quad c \quad c}{c} \end{aligned}$$

As the final rule of inference for  $BSA$  we have the induction inference.

$$(\text{ind}) \frac{[a_x[x]][a_x[y]] \quad [a_x[x]][a_x[y]]}{a_z[\text{nil}] \quad a_x[\text{cons}(x, y)] \quad a_x[\text{snoc}(x, y)]} \quad a_z[t]$$

The assumptions discharged by this rule are called *induction hypotheses*. In this rule, the variables  $x$  and  $y$  must be distinct and must satisfy the *eigen variables condition*. Namely, the variables  $x$  and  $y$  may not occur free in  $a_x[\text{nil}]$  or in any assumption other than the induction hypotheses on which the premises  $a_x[\text{cons}(x, y)]$  and  $a_x[\text{snoc}(x, y)]$  depend.

### 3.6. interpretation

We now explain the intended interpretation of the theory  $BSA$ . The intended domain of interpretation of our theory is  $S$ . We first define the denotation  $\llbracket t \rrbracket$  of a closed term  $t$  as follows.

1.  $\llbracket \text{nil} \rrbracket \equiv *$
2.  $\llbracket \text{cons}(s, t) \rrbracket \equiv [ \llbracket s \rrbracket . \llbracket t \rrbracket ]$
3.  $\llbracket \text{snoc}(s, t) \rrbracket \equiv ( \llbracket s \rrbracket . \llbracket t \rrbracket )$

It should be clear that each closed term denotes a unique sexp, and for each sexp  $t$  there uniquely exists a closed term  $t$  which denotes  $t$ .

We next assign a truth value (*true* or *false*) with each quantifier free closed wff. We first define the set of *descendants* of a sexp as follows.

1. The descendants of  $*$  is empty.
2. The descendants of  $[s . t]$  is the union of the descendants of  $s$  and  $t$  and the set  $\{s, t\}$ .
3. The descendants of  $(s . t)$  is empty.

Thus, for instance, the descendants of  $[[*].(*.*)]$  is the set  $\{*, [*, (*.*)\}$ . We say that  $s$  is a *descendant* of  $t$  if  $s$  is a member of the descendants of  $t$ .

Let  $s$  and  $t$  be closed terms and let  $s$  and  $t$  respectively be their denotations. Then the closed wff  $s=t$  is *true* if  $s$  and  $t$  are the same sexp, and it is *false* if  $s$  and  $t$  are distinct. The closed wff  $s < t$  is *true* if  $s$  is a descendant of  $t$  and is *false* otherwise.

Let  $a$  be any closed quantifier free wff. Since it is a propositional combination of the atomic wffs of the above form, we can calculate its truth value by first replacing each atomic sub-wff by its value and then evaluating the resulting boolean expression in the usual way.

We now define the class of *primitive wffs* for which we can also assign truth values if they are closed.

- 1–2. If  $s$  and  $t$  are terms then  $s=t$  and  $s < t$  are primitive wffs.
- 3–4. If  $a_1, \dots, a_n$  ( $n \geq 0$ ) are primitive wffs then  $a_1 \wedge \dots \wedge a_n$  and  $a_1 \vee \dots \vee a_n$  are primitive wffs.
5. If  $a_1, \dots, a_n$  ( $n \geq 0$ ) and  $b$  are primitive wffs then  $a_1, \dots, a_n \rightarrow b$  is a primitive wff.
- 6–7. If  $x_1, \dots, x_n$  is a sequence of distinct pure variables,  $t_1, \dots, t_n$  is a sequence of terms,  $u_i$  ( $1 \leq i \leq n$ ) is the elevation of  $t_i$  with respect to  $x_1, \dots, x_n$  and  $a$  is a primitive wff then  $\forall(x_1, \dots, x_n; x_1 < u_1,$

$\dots, x_n < w_n \rightarrow a$ ) and  $\exists(x_1, \dots, x_n; x_1 < w_1 \wedge \dots \wedge x_n < w_n \rightarrow a)$  are primitive wffs.

The primitive wffs defined by the clauses 6 and 7 above will respectively be abbreviated as:

$$\forall(x_1 < t_1, \dots, x_n < t_n; a)$$

$$\exists(x_1 < t_1, \dots, x_n < t_n; a)$$

(We will use this abbreviation for any wff  $a$  as well.) Since for each sexp  $t$  we can calculate the set of its descendants which is a finite set, it should be clear that we can uniquely assign a truth value for each closed primitive wff.

Next, we define  $\Sigma$ -wffs as follows:

1. A primitive wff is a  $\Sigma$ -wff.
- 2-3. If  $a_1, \dots, a_n$  ( $n \geq 0$ ) are  $\Sigma$ -wffs then  $a_1 \wedge \dots \wedge a_n$  and  $a_1 \vee \dots \vee a_n$  are  $\Sigma$ -wffs.
4. If  $a_1, \dots, a_n$  ( $n \geq 0$ ) are primitive wffs and  $b$  is a  $\Sigma$ -wff then  $a_1, \dots, a_n \rightarrow b$  is a  $\Sigma$ -wff.
5. If  $x_1, \dots, x_n$  is a sequence of distinct pure variables and  $a$  is a  $\Sigma$ -wff then  $\exists(x_1, \dots, x_n; a)$  is a  $\Sigma$ -wff.

We can define the *truth* of a closed  $\Sigma$ -wff inductively. The definition for the cases 1-4 is given similarly as for primitive wffs. For the case 5, we give the following definition. A closed  $\Sigma$ -wff  $\exists(x_1, \dots, x_n; a)$  is defined to be *true* if we can find a sequence of closed terms  $t_1, \dots, t_n$  for which  $a_{x_1, \dots, x_n}[\![t_1, \dots, t_n]\!]$  becomes *true*.

We may say that  $\mathcal{BSA}$  is *correct* if any closed  $\Sigma$ -wff which is provable in  $\mathcal{BSA}$  is *true*. In this paper we assume the correctness of  $\mathcal{BSA}$  without any further arguments. In particular we assume that  $\mathcal{BSA}$  is *consistent* in the sense that there is no proof of the wff  $\perp$ .

### 3.7. $\mathcal{BSA}$ as a formal system

We now define  $\mathcal{BSA}$  as a formal system and then define an isomorphism from  $\mathcal{BSA}$  to  $\mathcal{BSA}$ . It is possible to regard this isomorphism as an (symbolic) arithmetization of  $\mathcal{BSA}$ . Here we will not define the concept of proof in  $\mathcal{BSA}$  since we give a full description of  $\mathcal{BSA}$  as a formal axiom system in the next Section.

Let *Non\_member*, *Pure\_variable*, *Pure\_variable\_list*, *Variable*, *Term*, *Wff* and *Wff\_list* respectively be the following sexps.

```

+ non_member
  | x, [ ]
  | x, [y.X]
    - ne[x, y]
    - non_member[x, X]
  ;
+ pure_variable
  | (: var . t)
  ;
+ pure_variable_list
  | [ ]
  | [x.X]
    - pure_variable[x]
    - non_member[x, X]
    - pure_variable_list[X]
  ;
+ variable
  | x
    - pure_variable[x]
  | (: free . x)
    - variable[x]
  ;
+ term
  | *
  | x
    - variable[x]
  | [s.t]
    - term[s]
    - term[t]
  | (: snoc . [s, t])
    - term[s]
    - term[t]
  ;
+ wff
  | eq[s, t]
    - term[s]

```

```

    - term[t]
| lt[s, t]
    - term[s]
    - term[t]
| and[.A]
    - wff_list[A]
| or[.A]
    - wff_list[A]
| imply[A, b]
    - wff_list[A]
    - wff[b]
| all[(: abs. [X, a])]
    - pure_variable_list[X]
    - wff[a]
| ex[(: abs. [X, a])]
    - pure_variable_list[X]
    - wff[a]
;
+ wff_list
  | [ ]
  | [a . A]
    - wff[a]
    - wff_list[A]
;

```

Then the formal system:

$\mathbb{BSA}_0 \equiv [Ne, Non\_member, Pure\_variable, Pure\_variable\_list, Variable, Term, Wff, Wff\_list]$

defines basic concepts in  $\mathbb{BSA}$ . Thus, for instance, we say that (a sexp)  $a$  is a *wff* if  $\vdash_{\mathbb{BSA}_0} wff[a]$  holds.

**Example 3.2.**

$(: * . *)$  is a term since we have  $\vdash_{\mathbb{BSA}_0} term[(: * . *)]$ .  $\square$

In this way we can continue to give a complete definition of  $\mathbb{BSA}$  as a formal system. But as we said earlier we will not do so here because we will give a complete definition of  $\mathbb{BSA}$  in the next Section.

We now explain that the concepts which we defined formally here are es-

sententially the same as the corresponding concepts which we defined for  $\mathcal{BSA}$ . To this end we define a translation from syntactic objects like terms or wffs in  $\mathcal{BSA}$  into  $\mathcal{S}$ . We denote the translation of  $d$  by  $d^\dagger$ .

Terms in  $\mathcal{BSA}$  are translated as follows.

- 1.1.  $\text{var}_t^\dagger$  is  $(\text{var} . t)$ .
- 1.2.  $\text{free}(x)^\dagger$  is  $(\text{free} . x^\dagger)$ .
2.  $\text{nil}^\dagger$  is  $*$ .
- 3–4.  $\text{snoc}(s, t)^\dagger$  is  $[s^\dagger, t^\dagger]$  and  $\text{snoc}(s, t)^\dagger$  is  $(: s^\dagger, t^\dagger)$ .

The translation of wffs in  $\mathcal{BSA}$  is defined as follows.

- 1–2.  $\text{eq}(s, t)^\dagger$  is  $\text{eq}[s^\dagger, t^\dagger]$  and  $\text{lt}(s, t)^\dagger$  is  $\text{lt}[s^\dagger, t^\dagger]$ .
- 3–4.  $\text{and}(a_1, \dots, a_n)^\dagger$  is  $\text{and}[a_1^\dagger, \dots, a_n^\dagger]$  and  $\text{or}(a_1, \dots, a_n)^\dagger$  is  $\text{or}[a_1^\dagger, \dots, a_n^\dagger]$ .
5.  $\text{imply}((a_1, \dots, a_n), b)^\dagger$  is  $\text{imply}[[a_1^\dagger, \dots, a_n^\dagger], b^\dagger]$ .
- 6–7.  $\text{all}((x_1, \dots, x_n), a)^\dagger$  is  $\text{all}[\text{abs} . [[x_1^\dagger, \dots, x_n^\dagger], a^\dagger]]$  and  $\text{exist}((x_1, \dots, x_n), a)^\dagger$  is  $\text{ex}[\text{abs} . [[x_1^\dagger, \dots, x_n^\dagger], a^\dagger]]$ .

It is then easy to verify that this translation sends each syntactic entity in  $\mathcal{BSA}$  into corresponding entity in  $\mathcal{BSA}$ . Thus if  $a$  is a wff in the sense of  $\mathcal{BSA}$  then  $a^\dagger$  is a wff in  $\mathcal{BSA}$ , that is, we have  $\vdash_{\mathcal{BSA}_0} \text{wff}[a^\dagger]$ . Moreover for each wff  $a$  in  $\mathcal{BSA}$  we can uniquely find a wff  $a$  in  $\mathcal{BSA}$  such that  $a^\dagger$  is  $a$ . A similar correspondence holds also for terms. It is also obvious from our definition that the translation is homomorphic with respect to the inductive definition of syntactic entities. We may thus conclude that both  $\mathcal{BSA}$  and  $\mathcal{BSA}$  give definitions to the abstract concepts such as terms or wffs in terms of their respective representations. For this reason we will use the same abbreviations which we used for syntactic entities in  $\mathcal{BSA}$  as abbreviations for the corresponding objects in  $\mathcal{BSA}$ . We will also use syntactic variables to make our intention clear. Thus for instance if in some context we wish to refer a certain  $\text{sexp}$  as a wff, we will use syntactic variables  $a$ ,  $b$  or  $c$  for it.

**Example 3.3.**

$\forall(x; \exists(x; x = \#x))$  is an abbreviation of the  $\text{sexp}$

$$\text{all}[(\text{abs} . [[x], \text{ex}[(\text{abs} . [[x], \text{eq}[x, (\text{free} . x)]]])]])]$$

which is a wff in  $\mathcal{BSA}$ .  $\square$



## §4. First Order Theories

In this section we introduce the formal system FOT (for *First Order Theory*). In FOT one can define a general class of first order theories including BSA and its extensions or restrictions. The basic idea is that any sexp  $S$  when viewed as a formal system can be used to define a formal theory in countable first-order language. We first introduce the concept of quasi-quotation which provides a convenient notation for various syntactic entities. We then define FOT using this notation.

## 4.1. quasi-quotation

When one makes a statement about an object one must use a name for that object to refer to that object. If that object is linguistic it is possible to use that object directly as the name of that object and such usage of linguistic object is called *autonomous*. Autonomous usage of an object as the name for it is harmless provided the symbols appearing in that linguistic object are not included in the vocabulary of the meta language, that is the language in which the statement is made. Another method of obtaining a name for a linguistic object which is applicable to most cases, is to *quote* that linguistic object. (Still, there is a problem of how to quote quotation marks. This problem is solved by computer scientists in many ways.)

The mechanism of quotation is so restrictive, and it is often desirable to be able to unquote part of the quotation. Quasi-quotation is such a mechanism and Quine used it in his book [9] extensively. Let us give an example of quasi-quotation. The fourth clause of our inductive definition of terms in 3.3 is as follows.

4. If  $s$  and  $t$  are terms then  $\text{smoc}(s, t)$  is a term.

Using corners which are Quine's notational device for quasi-quotation, the above clause will look like this:

4. If  $s$  and  $t$  are terms then  $\lceil \text{smoc}(s, t) \rceil$  is a term.

The difference of quasi-quotation from ordinary quotation is this. Within quasi-quotation marks (corners in this case) one can embed meta variables (' $s$ ' and ' $t$ ' in this case) among the symbols of the object language. Such a quasi-quotation designates an expression that result from the content of the quasi-quotation by replacing each meta variable by what it designates. Thus if both

$s$  and  $t$  designates 'mil' then  $\lceil \text{cons}(s, t) \rceil$  designates the expression 'cons(mil, mil)'. This is the basic idea of quasi-quotation. However, with a slight danger of falling into the trap of *use* and *mention* confusion (Quine [9], pp. 23–26.), we have avoided the use of quasi-quotation nor quotation for the ease of readability.

Quasi-quotation is a familiar idea among users of computers who use editors, text-formatters etc. (See, e.g., Bourne [2].) Some LISP languages like Maclisp have a mechanism of quasi-quotation as an input macro known as *back quote macro*. For example, if the variables S and T both has NIL as its current value then evaluating the expression:

```
`(, S . , T)
```

will return the value (NIL . NIL). Since in LISP the same symbols are used both in the meta language and the object language, a comma is placed within the scope of back quote to tell the input routine that the following expression should be evaluated (rather than quoted).

We will introduce Maclisp like quasi-quotation mechanism as a convenient notational device for referring to various syntactic objects within a formal system. In connection with this, we note that we have incorporated the mechanism of quotation as a part of evaluation mechanism. (Recall (E6) of the definition of *eval*.) We define our quasi-quotation mechanism formally by the formal system  $[Quasi\_quote, Ne]$ , where *Quasi\_quote* is the following sexp and *Ne* was defined in  $\mathbb{U}niv$ :

```
+ quasi_quote
  | *, *
  | [s . t], [u . v]
    - quasi_quote[s, u]
    - quasi_quote[t, v]
  | (: s . t), (: snoc . [u, v])
    - ne[s, *]
    - ne[s, eval]
    - quasi_quote[s, u]
    - quasi_quote[t, v]
  | (: * . t), (: * . t)
  | (: eval . t), t
;
```

We can easily verify that for any sexp  $t$  there uniquely exists a sexp  $v$  such that:

$$\vdash_{[Quasi\_quote, Ne]} \text{quasi\_quote}[t, v]$$

In this case we call  $v$  the *quasi-quote expansion* of  $t$ . We introduce the following abbreviations.

$\backslash t$  for the quasi-quote expansion of  $t$   
 $/t$  for (eval.  $t$ )

**Example 4.1.**

$$\begin{aligned} \backslash (: /s. /t) &\equiv \backslash (\text{snoc. } [ /s, /t]) \\ &\equiv (\text{snoc. } (\backslash \text{snoc. } \backslash [ /s, /t])) \\ &\equiv (\text{snoc. } (\text{snoc. } [ \backslash /s, \backslash /t])) \\ &\equiv (\text{snoc. } (\text{snoc. } [s, t])) \\ &\equiv (: \text{snoc. } [s, t]) \quad \square \end{aligned}$$

We can therefore rewrite the sexp *Term* in the formal system  $\text{BSA}_0$  as follows.

```
+ term
  | *
  | x
    - variable[x]
  | [s. t]
    - term[s]
    - term[t]
  | \(: /s. /t)
    - term[s]
    - term[t]
;
```

The reader should compare this with the definition of terms in 3.3 and consider the reason why quasi-quotation is not necessary for *cons* in this formal definition of terms.  $\square$

*Note.* We have introduced the mechanism of quasi-quotation as a part of evaluation mechanism in our former programming languages for the domain  $S_{old}$ . (See [10], [11].) Edinburgh LCF [6] also uses quasi-quotation as convenient notation for its PP $\lambda$  objects.  $\square$

## 4.2. FOT

We define the formal system **FOT** as a list of 39 sexps of which first 7 are taken from the formal system **Univ** and the remaining sexps are the following 32 sexps.

```

+ member
  | x, [x.X]
  | x, [y.X]
    - member[x, X]
  ;
+ non_member
  | x, [ ]
  | x, [y.X]
    - ne[x, y]
    - non_member[x, X]
  ;
+ pure_variable
  | (: var.t)
  ;
+ pure_variable_list
  | [ ]
  | [x.X]
    - pure_variable[x]
    - non_member[x, X]
    - pure_variable_list[X]
  ;
+ variable
  | x
    - pure_variable[x]
  | (: free.x)
    - variable[x]
  ;
+ pure_part
  | (: var.t), (: var.t)
  | (: free.x), y

```

```

    - pure_part[x, y]
;
+ length
  | *, *
  | [x.X], [* . n]
    - length[X, n]
;
+ symbol
  | (: * . t)
;
+ term
  | x, S
    - variable[x]
  | *, S
    - theorem[special[nil], S]
  | [s . t], S
    - theorem[special[cons], S]
    - term[s, S]
    - term[t, S]
  | `(: /s . /t), S
    - theorem[special[snoc], S]
    - term[s, S]
    - term[t, S]
  | (: apply . [Fun . T]), S
    - symbol[Fun]
    - term_list[T, S]
    - length[T, Arity]
    - theorem[function[[Fun . Arity]], S]
  | (: quote . t), S
    - theorem[special[quote], S]
  | (: * . t), S
    - theorem[special[symbol], S]
;
+ term_list
  | [ ], S
  | [t . T], S

```

```

        - term[t, S]
        - term_list[T, S]
    ;
+ admissible
  | p
    - symbol[p]
    - ne[p, and]
    - ne[p, or]
    - ne[p, imply]
    - ne[p, all]
    - ne[p, ex]
  ;
+ wff
  | eq[s, t], S
    - term[s, S]
    - term[t, S]
  | [Prd . T], S
    - admissible[Prd]
    - term_list[T, S]
    - length[T, Arity]
    - theorem[predicate[[Prd . Arity]], S]
  | and[. A], S
    - wff_list[A, S]
  | or[. A], S
    - wff_list[A, S]
  | imply[A, b], S
    - wff_list[A, S]
    - wff[b, S]
  | all[: abs . [X, a]], S
    - pure_variable_list[X]
    - wff[a, S]
  | ex[: abs . [X, a]], S
    - pure_variable_list[X]
    - wff[a, S]
  ;
+ wff_list

```

```

| [ ], S
| [a . A], S
    - wff[a, S]
    - wff_list[A, S]
;
+ find
| [ ], [ ], x, x
| [t . T], [x . X], x, t
| [t . T], [y . X], x, v
    - ne[x, y]
    - find[T, X, x, v]
;
+ rename
| x, y, x
    - pure_variable[x]
| (: free . x), y, x
    - pure_variable[y]
| (: free . x), (: free . y), (: free . z)
    - rename[x, y, z]
;
+ free
| *, X, *
| y, X, (: free . y)
    - variable[y]
    - pure_part[y, x]
    - member[x, X]
| y, X, y
    - variable[y]
    - pure_part[y, x]
    - non_member[x, X]
| [s . t], X, [u . v]
    - free[s, X, u]
    - free[t, X, v]
| `(: /s . /t), X, `(: /u . /v)
    - free[s, X, u]
    - free[t, X, v]

```

```

| (: *. t), X, (: *. t)
| (: quote . t), X, (: quote . t)
| (: apply . [f. T]), X, (: apply . [f. V])
    - free[T, X, V]
| (: eval . t), X, (: eval . t)
;
+ subst
| T, X, x, v
    - variable[x]
    - find[T, X, x, v]
| T, X, *, *
| T, X, [s. t], [u. v]
    - subst[T, X, s, u]
    - subst[T, X, t, v]
| T, X, `(: /s. /t), `(: /u. /v)
    - subst[T, X, s, u]
    - subst[T, X, t, v]
| T, X, (: *. t), (: *. t)
| T, X, (: quote . t), (: quote . t)
| T, X, (: apply . [f. U]), (: apply . [f. V])
    - subst[T, X, U, V]
| T, X, (: abs . [Y, t]), (: abs . [Y, v])
    - pure_variable_list[Y]
    - free[T, Y, V]
    - free[X, Y, Z]
    - subst[V, Z, t, v]
;
+ subst 1
| t, x, s, v
    - subst[[t], [x], s, v]
;
+ bind_find
| [ ], [ ], x, x
| [t. T], [x. X], x, t
| [t. T], [y. X], x, v

```



```

    - ne[x, y]
    - pure_part[x, u]
    - pure_part[y, u]
    - rename[x, y, v]
| [t. T], [y. X], x, v
    - ne[x, y]
    - pure_part[x, u1]
    - pure_part[y, u2]
    - ne[u1, u2]
    - bind_find[T, X, x, v]
;
+ bind_subst
  | T, X, x, v
    - variable[x]
    - bind_find[T, X, x, v]
  | T, X, *, *
  | T, X, [s. t], [u. v]
    - bind_subst[T, X, s, u]
    - bind_subst[T, X, t, v]
  | T, X, `(: /s. /t), `(: /u, /v)
    - bind_subst[T, X, s, u]
    - bind_subst[T, X, t, v]
  | T, X, t, t
    - symbol[t]
  | T, X, (: quote . t), (: quote . t)
  | T, X, (: apply . [f. U]), (: apply [f. V])
    - bind_subst[T, X, U, V]
  | T, X, (: abs . [Y, t]), (: abs . [Y, v])
    - pure_variable_list[Y]
    - free [T, Y, V]
    - free[X, Y, Z]
    - bind_subst[V, Z, t, v]
;
+ append
  | [ ], Y, Y
  | [x. X], Y, [x. Z]

```

```

        - append[X, Y, Z]
    ;
+ addend
  | [ ], Y, Y
  | [x.X], Y, Z
    - addend[X, Y, Z]
    - member[x, Z]
  | [x.X], Y, [x.Z]
    - addend[X, Y, Z]
    - non_member[x, Z]
  ;
+ vars
  | x, [x]
    - variable[x]
  | *, [ ]
  | [s.t], V
    - vars[s, S]
    - vars[t, T]
    - addend[S, T, V]
  | `(: /s./t), V
    - vars[s, S]
    - vars[t, T]
    - addend[S, T, V]
  | (: *.t), [ ]
  | (: quote.t), [ ]
  | (: apply.[f.T]), V
    - vars[T, V]
  | (: abs.[X, a]), V
    - vars[a, U]
    - down[X, U, V]
  ;
+ down
  | X, [ ], [ ]
  | X, [x.Ū], V
    - pure_variable[x]
    - member[x, X]

```

```

    - down[X, U, V]
| X, [x. U], [x. V]
    - pure_variable[x]
    - non_member[x, X]
    - down[X, U, V]
| X, [( : free . y ) . U], [y . V]
    - pure_part[y, z]
    - member[z, X]
    - down[X, U, V]
| X, [( : free . y ) . U], [( : free . y ) . V]
    - pure_part[y, z]
    - non_member[z, X]
    - down[X, U, V]
;
+ new
  | x, t
    - vars[t, X]
    - non_member[x, X]
;
+ new_list
  | *, t
  | [x. X], t
    - new[x, t]
    - new_list[X, t]
;
+ eq_pr
  | S, E, [ ], [ ], [ ]
  | S, E, [p. P], [u. U], [v. V]
    - pr[S, E, p, u=v]
    - eq_pr[S, E, P, U, V]
;
+ lpr
  | S, E, [ ], [ ]
  | S, E, [p. P], [a. A]
    - pr[S, E, p, a]
    - lpr[S, E, P, A]

```

```

;
+ upr
  | S, [ ], E, [ ], c
  | S, [a . A], E, [p . P], c
    - pr[S, [a . E], p, c]
    - upr[S, A, E, P, c]
;
+ axiom
  | a, S
    - proves[p, axiom[a], S]
    - wff[a, S]
;
+ pr
  | S, E, [a, axiom[ ]], a
    - axiom[a, S]
    - wff_list[E, S]
  | S, E, [a, assumption[ ]], a
    - member[a, E]
    - wff_list[E, S]
  | S, E, [and[ . A], and_I[ . P]], and [ . A]
    - lpr[S, E, P, A]
  | S, E, [a, and_E[p]], a
    - pr[S, E, p, and[ . A]]
    - member[a, A]
  | S, E, [or[ . A], or_I[p]], or[ . A]
    - pr[S, E, p, a]
    - member[a, A]
    - wff_list[A, S]
  | S, E, [a, or_E[p . P]], a
    - pr[S, E, p, or[ . A]]
    - upr[S, A, E, P, a]
  | S, E, [imply[A, b], imply_I[p]], imply[A, b]
    - append[A, E, F]
    - pr[S, F, p, b]
  | S, E, [a, imply_E[p, P]], a
    - pr[S, E, p, imply[A, a]]

```

```

    - lpr[S, E, P, A]
| S, E, [all[: abs. [X, a]], all_I[[X, Y], p]], all[: abs. [X, a]]]
    - new_list[Y, E]
    - new_list[Y, (: abs. [X, a])]
    - pure_variable_list[X]
    - wff[a, S]
    - bind_subst[Y, X, a, b]
    - pr[S, E, p, b]
| S, E, [a, all_E[T, p]], a
    - pr[S, E, p, all[: abs. [X, b]]]
    - term_list[T, S]
    - bind_subst[T, X, b, a]
| S, E, [ex[: abs. [X, b]], ex_I[T, p]], ex[: abs. [X, b]]]
    - term_list[T, S]
    - pure_variable_list[X]
    - wff[b, S]
    - bind_subst[T, X, b, a]
    - pr[S, E, p, a]
| S, E, [a, ex_E[Y, p, q]], a
    - new_list[Y, E]
    - new_list[Y, a]
    - pr[S, E, p, ex[: abs. [X, b]]]
    - bind_subst[Y, X, b, c]
    - pr[S, [c. E], q, a]
| S, E, [t=t, axiom_id[ ]], t=t
    - term[t, S]
| S, E, [a, subst[[U, V, X, b], p. P]], a
    - wff[b, S]
    - subst[U, X, b, c]
    - subst[V, X, b, a]
    - pr[S, E, p, c]
    - eq_pr[S, E, P, U, V]
;
+ thm
  | a, S
    - pr[S, [ ], p, a]

```

;

Let us briefly explain the intended meanings of some important concepts defined in FOT. We will call any formal system an *axiom system*. Then term[ $t, S$ ] means that  $t$  is a *term* in the axiom system  $S$ . Similarly, wff[ $a, S$ ] means that  $a$  is a *wff* in  $S$ . Note that here  $S$  is used to specify the language. The operation of substitution is defined by the predicate ‘subst’. Namely, subst[ $T, X, t, v$ ] means that  $v$  is the result of simultaneously substituting members of the list  $T$  for the free occurrences of the corresponding members of the list  $X$  in  $t$ . The uniqueness of the result of substitution is guaranteed by the following easily verifiable fact:

$$\vdash_{\text{FOT}} \text{subst}[T, X, t, v_1], \vdash_{\text{FOT}} \text{subst}[T, X, t, v_2] \implies v_1 \equiv v_2$$

Let  $T \equiv [t_1, \dots, t_n]$  and  $X \equiv [x_1, \dots, x_n]$ . If  $\vdash_{\text{FOT}} \text{subst}[T, X, t, v]$  holds for some  $v$  then such a  $v$  is unique by the above fact. We will denote this  $v$  by:

$$t_{x_1, \dots, x_n}[t_1, \dots, t_n]$$

The meaning of vars[ $t, X$ ] is that  $X$  is the list of variables occurring free in  $t$ . We will say that  $x$  is *new* to  $t$  if new[ $x, t$ ] holds where new[ $x, t$ ] means that  $x$  does not occur free in  $t$ . Finally, pr[ $S, E, p, a$ ] means that  $p$  is a *proof* in the axiom system  $S$  of  $a$  from the assumption  $E$ .

It should be clear that arbitrary formal theory with countable first-order language over intuitionistic logic (or classical logic) can be treated in this framework. As the first concrete example of an axiom system we now define the axiom system BSA as follows:

BSA  $\equiv$  [Special, Predicate, Axiom, Bas\_term, Bsa\_wff, Bsa\_wff\_list, Variable, Pure\_variable, Pure\_variable\_list, Non\_member, Ne, New, Vars, Addend, Member, Subst, Find, Free, Pure\_part, SubstI]

where we have defined the last 14 sexps already and the first 6 sexps are respectively as follows.

+ special  
 | nil  
 | cons  
 | snoc  
 ;  
 + predicate

```

| lt[* , *]
;
+ axiom
| '∀(s, t; [s.t] ≠ *)
| '∀(s, t; (: s.t) ≠ *)
| '∀(s, t, u, v; [s.t] ≠ (: u.v))
| '∀(s, t, u, v; [s.t] = [u.v] → s = u)
| '∀(s, t, u, v; [s.t] = [u.v] → t = v)
| '∀(s, t, u, v; (: s.t) = (: u.v) → s = u)
| '∀(s, t, u, v; (: s.t) = (: u.v) → t = v)
| '∀(r; ¬(r < *))
| '∀(r, s, t; ¬(r < (: s.t)))
| '∀(s, t; s < [s.t])
| '∀(s, t; t < [s.t])
| '∀(r, s, t; r < s → r < [s.t])
| '∀(r, s, t; r < t → r < [s, t])
| '∀(r, s, t; r < [s.t] → r = s ∨ r < s ∨ r = t ∨ r < t)
| '(/a0, ∀(x, /y; /IHx, /IHy → /a1), ∀(x, /y; /IHx, /IHy → /a2)
→ /b)
    - pure_variable[x]
    - pure_variable[y]
    - pure_variable[z]
    - ne[x, y]
    - bsa_wff[a]
    - new[x, a]
    - new[y, a]
    - bas_term[t]
    - subst[t, z, a, b]
    - subst[* , z, a, a0]
    - subst[x, z, a, IHx]
    - subst[y, z, a, IHy]
    - subst[[x.y], z, a, a1]
    - subst['(: /x./y), z, a, a2]
+ bsa_term
| x
    - variable[x]

```

```

| *
| [s. t]
    - bsa_term[s]
    - bsa_term[t]
| `(: /s. /t)
    - bsa_term[s]
    - bsa_term[t]
;
+ bsa_wff
| eq[s, t]
    - bsa_term[s]
    - bsa_term[t]
| lt[s, t]
    - bsa_term[s]
    - bsa_term[t]
| and[. A]
    - bsa_wff_list[A]
| or[. A]
    - bsa_wff_list[A]
| imply[A, b]
    - bsa_wff_list[A]
    - bsa_wff[b]
| all[( : abs. [X, a])]
    - pure_variable_list[X]
    - bsa_wff[a]
| ex[( : abs. [X, a])]
    - pure_variable_list[X]
    - bsa_wff[a]
;
+ bsa_wff_list
| [ ]
| [a. A]
    - bsa_wff[a]
    - bsa_wff_list[A]
;

```

We now introduce notations for some concepts which we defined in FOT.



Let  $S$  be an axiom system. We will say that  $p$  is a *proof* of  $a$  from the *assumptions*  $E$  in  $S$  if

$$\vdash_{\text{FOTPr}}[S, E, p, a]$$

holds, and we use the notation

$$S: p \vdash a \text{ for } \vdash_{\text{FOTPr}}[S, [ ], p, a]$$

Similarly we will say that  $a$  is a *theorem* of  $S$  if

$$S: p \vdash a$$

holds for some  $p$  and use the notation

$$S \vdash a$$

for it.

We are now using the provability sign ‘ $\vdash$ ’ for two purposes. Namely, one usage is for the provability in formal systems and the other is for the provability in axiom systems. However, these usages can always be distinguished syntactically by the presence or absence of a subscript for the provability sign.

Let  $S$  and  $T$  be axiom systems. We say that  $T$  is an *extension* of  $S$  if for any sexp  $a$  we have:

- (i) if  $\vdash_{\text{FOT}} \text{wff}[a, S]$  then  $\vdash_{\text{FOT}} \text{wff}[a, T]$  and
- (ii) if  $S \vdash a$  then  $T \vdash a$ .

(The condition (i) is, in fact, redundant since it follows from (ii).)  $T$  is said to be a *conservative extension* of  $S$  if:

- (i)  $T$  is an extension of  $S$  and
- (ii) if  $\vdash_{\text{FOT}} \text{wff}[a, S]$  and  $T \vdash a$  then  $S \vdash a$ .

$S$  is said to be *consistent* if for no  $p$

$$S: p \vdash \perp$$

holds. It is clear that if  $S$  is consistent and  $T$  is a conservative extension of  $S$  then  $T$  is also consistent.

## §5. The Axiom System SA

In this Section we introduce an axiom system SA which is a conservative extension of BSA. The motivation for the extension of the system is to obtain a system which is powerful enough to let one work actually within the system

comfortably. Our final goal is to get a system in which one can formally carry out all the mathematical and metamathematical arguments we are informally doing in this paper. The introduction of SA is a first step toward this goal.

### 5.1. SA

The axiom system SA is defined as the sexp:

$$\text{SA} \equiv [\textit{Special}, \textit{Predicate}, \textit{Axiom}, \textit{Sa\_term}, \textit{Sa\_wff}, \textit{Sa\_wff\_list}, \\ \textit{Variable}, \textit{Pure\_variable}, \textit{Pure\_variable\_list}, \textit{Non\_member}, \textit{Ne}, \\ \textit{New}, \textit{Vars}, \textit{Addend}, \textit{Member}, \textit{Subst}, \textit{Find}, \textit{Free}, \textit{Prue\_part}, \\ \textit{SubstII}]$$

where the last 14 members of SA are already defined elsewhere and the first 6 are listed below. The number enclosed between ‘%’ symbols are comments, and we will use this number as the number of the axiom defined by the corresponding clause.

```
+ special
  | nil
  | cons
  | snoc
  | quote
  | symbol
  ;
+ predicate
  | lt[* , *]
  ;
+ axiom
  | '∀(s, t; [s. t] ≠ *) % 1 %
  | '∀(s, t; (: s. t) ≠ *) % 2 %
  | '∀(s, t, u, v; [s. t] ≠ (: u. v)) % 3 %
  | '∀(s, t, u, v; [s. t] = [u. v] → s = u) % 4 %
  | '∀(s, t, u, v; [s. t] = [u. v] → t = v) % 5 %
  | '∀(s, t, u, v; (: s. t) = (: u. v) → s = u) % 6 %
  | '∀(s, t, u, v; (: s. t) = (: u. v) → t = v) % 7 %
  | '∀(r; ¬(r < *)) % 8 %
  | '∀(r, s, t; ¬(r < (: s. t))) % 9 %
  | '∀(s, t; s < [s. t]) % 10 %
```

|  $\forall(s, t; t < [s.t])$  % 11 %  
 |  $\forall(r, s, t; r < s \rightarrow r < [s.t])$  % 12 %  
 |  $\forall(r, s, t; r < t \rightarrow r < [s.t])$  % 13 %  
 |  $\forall(r, s, t; r < [s.t] \rightarrow r = s \vee r < s \vee r = t \vee r < t)$  % 14 %  
 |  $((\text{quote}.*) = *)$  % 15 %  
 |  $((\text{quote}.[s./t]) = [(\text{quote}./s).(\text{quote}./t)])$  % 16 %  
 |  $((\text{quote}.(/s./t)) = (:(\text{quote}./s).(\text{quote}./t)))$  % 17 %  
 |  $((*.*) = (:*.*))$  % 18 %  
 |  $((*./s) = (:*./u), (*./t) = (:*./v) \rightarrow$   
      $(*.[s./t]) = (:*.[u./v]))$  % 19 %  
 |  $((*./s) = (:*./u), (*./t) = (:*./v) \rightarrow$   
      $(*(./s./t)) = (:*(./u./v)))$  % 20 %  
 |  $(/a0, \forall(/x./y; /IHx, /IHy \rightarrow /a1),$   
      $\forall(/x./y; /IHx, /IHy \rightarrow /a2) \rightarrow /b)$  % 21 %  
     – pure\_variable[x]  
     – pure\_variable[y]  
     – pure\_variable[z]  
     – ne[x, y]  
     – sa\_wff[a]  
     – new[x, a]  
     – new[y, a]  
     – sa\_term[t]  
     – subst[t, z, a, b]  
     – subst[\* , z, a, a0]  
     – subst[x, z, a, IHx]  
     – subst[y, z, a, IHy]  
     – subst[[x.y], z, a, a1]  
     – subst[( : /x./y), z, a, a2]  
 |  $(\forall(/z; \forall(/w; /w < /z \rightarrow /a1) \rightarrow /a) \rightarrow /b)$  % 22 %  
     – pure\_variable[z]  
     – pure\_variable[w]  
     – ne[z, w]  
     – sa\_wff[a]  
     – new[w, a]  
     – sa\_term[t]  
     – subst[w, z, a, a1]

```

      - substl[t, z, a, b]
    ;
+ sa_term
  | x
    - variable[x]
  | *
  | [s . t]
    - sa_term[s]
    - sa_term[t]
  | `(: /s. /t)
    - sa_term[s]
    - sa_term[t]
  | (: * . t)
  | (: quote . t)
  ;
+ sa_wff
  | eq[s, t]
    - sa_term[s]
    - sa_term[t]
  | lt[s, t]
    - sa_term[s]
    - sa_term[t]
  | and[. A]
    - sa_wff_list[A]
  | or[. A]
    - sa_wff_list[A]
  | imply[A, b]
    - sa_wff_list[A]
    - sa_wff[b]
  | all[( : abs . [X, a])]
    - pure_variable_list[X]
    - sa_wff[a]
  | ex[( : abs . [X, a])]
    - pure_variable_list[X]
    - sa_wff[a]
  ;

```

$$\begin{array}{l}
+ \text{ sa\_wff\_list} \\
\quad | [ ] \\
\quad | [a . A] \\
\quad \quad - \text{ sa\_wff}[a] \\
\quad \quad - \text{ sa\_wff\_list}[A] \\
\quad ;
\end{array}$$

Axiom 22 is the induction schema with respect to the ordering  $<$  and we will refer to it as the  $<$ -induction

## 5.2. SA is conservative over BSA

In this subsection we show that SA is conservative over BSA. To prove this we consider an intermediate axiom system  $\text{SA}_0$  which results from SA by simply deleting the  $<$ -induction schema (that is, axiom 22). We first show that  $\text{SA}_0$  is conservative over BSA and then prove that SA is conservative over  $\text{SA}_0$ .

We prove the first part in the following form. We first define a mapping  $(-)^*$  which sends a wff in  $\text{SA}_0$  to a wff in BSA and then prove that this mapping has the following properties which are sufficient to prove that  $\text{SA}_0$  is conservative over BSA.

- (i) If  $\text{SA}_0: p \vdash a$  then  $\text{BSA}: q \vdash a^*$  for some  $q$ .
- (ii) If  $a$  is a wff in BSA then  $a^*$  is  $a$ .

(We will use  $p$  and  $q$  as syntactic variables for proofs.)

We first translate terms in  $\text{SA}_0$  to terms in BSA. The translation sends variables to the same variables,  $*$  to  $*$  and is homomorphic with respect to *cons* and *snoc*. The rest of the translation is defined as follows.

- 5.1.  $(\text{quote } . *)^* \equiv *$
- 5.2.  $(\text{quote } . [s . t])^* \equiv [(\text{quote } . s)^* . (\text{quote } . t)^*]$
- 5.3.  $(\text{quote } . (s . t))^* \equiv (: (\text{quote } . s)^* . (\text{quote } . t)^*)$
- 6.1.  $(* . *)^* \equiv (: * . *)$
- 6.2. If  $(* . s)^* \equiv (: * . w)$  and  $(* . t)^* \equiv (: * . v)$   
then  $(* . [s . t])^* \equiv (: * . [w . v])$ .
- 6.3. If  $(* . s)^* \equiv (: * . w)$  and  $(* . t)^* \equiv (: * . v)$   
then  $(* . (s . t))^* \equiv (: * . (: w . v))$ .

We then extend  $(-)^*$  homomorphically to wffs. It is easy to see that if  $a$  is a wff in  $\text{SA}_0$  then  $a^*$  is a wff in BSA and that this mapping has the property (ii) above. We now prove (i) by the (informal)  $<$ -induction on  $p$ . More precisely,

we prove the following proposition by the  $<$ -induction on  $p$ .

$$\begin{aligned}
& \vdash_{\text{FOT}} \text{pr}[\text{SA}_0, E, p, a] \\
& \implies \vdash_{\text{FOT}} \text{pr}[\text{BSA}, E^*, q, a^*] \text{ for some } q \text{ and} \\
& \vdash_{\text{FOT}} \text{lpr}[\text{SA}_0, E, p, a] \\
& \implies \vdash_{\text{FOT}} \text{lpr}[\text{BSA}, E^*, q, a^*] \text{ for some } q \text{ and} \\
& \vdash_{\text{FOT}} \text{upr}[\text{SA}_0, A, E, p, a] \\
& \implies \vdash_{\text{FOT}} \text{upr}[\text{BSA}, A^*, E^*, q, a^*] \text{ for some } q \text{ and} \\
& \vdash_{\text{FOT}} \text{eq\_pr}[\text{SA}_0, E, p, U, V] \\
& \implies \vdash_{\text{FOT}} \text{eq\_pr}[\text{BSA}, E^*, q, U^*, V^*] \text{ for some } q.
\end{aligned}$$

In proving this proposition we may assume the proposition for any  $\text{sexp}$  which is a descendant of  $p$ . Since this proposition is a conjunction of four propositions we must prove each of them. To prove the first proposition let us assume:

$$\vdash_{\text{FOT}} \text{pr}[\text{SA}_0, E, p, a]$$

Then according to the definition of  $\text{pr}$  in  $\text{FOT}$  we have 14 cases to consider.

(Case 1) In this case we have for some  $p_0$ :

$$p \equiv [a, \text{axiom}[p_0]] \quad (1.1)$$

$$\vdash_{\text{FOT}} \text{proves}[p_0, \text{axiom}[a], \text{SA}_0] \quad (1.2)$$

$$\vdash_{\text{FOT}} \text{wff}[a, \text{SA}_0] \quad (1.3)$$

and

$$\vdash_{\text{FOT}} \text{wff\_list}[E, \text{SA}_0] \quad (1.4)$$

Since Theorem 2.1 still holds even if we substitute  $\text{FOT}$  for  $\text{U}_{\text{univ}}$  in it, we have by (1.3):

$$p_0 \vdash_{\text{SA}_0} \text{axiom}[a] \quad (1.5)$$

We will prove that we can either find a  $q_0$  for which we have

$$q_0 \vdash_{\text{BSA}} \text{axiom}[a^*] \quad (1.6)$$

or find a  $q_0$  for which we have

$$\vdash_{\text{FOT}} \text{pr}[\text{BSA}, E^*, q_0, a^*] \quad (1.7)$$

According to the definition of  $\text{axiom}$  in  $\text{SA}_0$  we have 21 cases to consider. For the first 14 cases,  $a^*$  is identical to  $a$  and we may take  $q_0$  as  $p_0$  to get (1.6). We next consider the remaining cases.

(Case 1.15) In this case  $a$  is  $((\text{quote } *) = *)$  and  $a^*$  is  $(* = *)$ . Putting  $q_0$  as

$[* = *, \text{axiom\_id}[ \ ]]$  we get (1.7).

(Case 1.16) In this case  $a$  is  $((\text{quote} . [s . t]) = [(\text{quote} . s) . (\text{quote} . t)])$  for some  $s$  and  $t$  and  $a^*$  is  $(T = T)$  where  $T$  is  $[(\text{quote} . s) . (\text{quote} . t)]$ . Putting  $q_0$  as  $[T = T, \text{axiom\_id}[ \ ]]$  we get (1.7).

(Case 1.17) is similar to (Case 1.16) and we omit it.

(Case 1.18) is similar to (Case 1.15) and we omit it.

(Case 1.19) In this case  $a$  is:

$$(* . s) = (: * . u), (* . t) = (: * . v) \rightarrow (* . [s . t]) = (: * . [u . v])$$

and  $a^*$  is:

$$(* . s)^* = (: * . u), (* . t)^* = (: * . v) \rightarrow (* . [s . t])^* = (: * . [u . v])$$

By the definition of  $(-)^*$  we can find  $S$  and  $T$  such that:

$$(* . s)^* \equiv (: * . S) \text{ and } (* . t)^* \equiv (: * . T)$$

We can now rewrite  $a^*$  as:

$$(: * . S) = (: * . u), (: * . T) = (: * . v) \rightarrow (: * . [S . T]) = (: * . [u . v])$$

Using axioms on equality we can construct a  $q_0$  for which we have (1.7).

(Case 1.20) is similar to (Case 1.19) and we omit it.

(Case 1.21) In this case  $a$  is:

$$a_0, \forall(x, y; IHx, IHy \rightarrow a_1), \forall(x, y; IHx, IHy \rightarrow a_2) \rightarrow b$$

and  $a^*$  is:

$$a_0^*, \forall(x, y; IHx^*, IHy^* \rightarrow a_1^*), \forall(x, y; IHx^*, IHy^* \rightarrow a_2^*) \rightarrow b^*$$

Then by modifying  $p_0$  suitably we can construct a  $q_0$  for which we have (1.6).

(We omit the details of the construction.)

We thus found a  $q_0$  for which we have (1.6) or (1.7). In case we have (1.7) we are done, so let us assume that we have (1.6). Then by Theorem 2.1 (with FOT replacing  $\text{U}_{\text{Miv}}$ ) we obtain:

$$\vdash_{\text{FOT}} \text{proves}[q_0, \text{axiom}[a], \text{BSA}] \quad (1.8)$$

Since  $a$  is a wff in  $\text{SA}_0$  and  $E$  is a wff\_list in  $\text{SA}_0$  we have:

$$\vdash_{\text{FOT}} \text{wff}[a^*, \text{BSA}] \quad (1.9)$$

$$\vdash_{\text{FOT}} \text{wff\_list}[E^*, \text{BSA}] \quad (1.10)$$

By (1.8)-(1.10) we have:

$$\vdash_{\text{FOT}} \text{pr}[\text{BSA}, E^*, [a^*, \text{axiom}[q_0]], a^*]$$

This completes the proof of the proposition for (Case 1).

(Case 2) is easy and we omit it.

(Case 3) In this case we have for some  $P$  and  $A$ :

$$p \equiv [a, \text{and\_I}[\cdot P]] \quad (3.1)$$

$$a \equiv \text{and}[\cdot A] \quad (3.2)$$

and

$$\vdash_{\text{FOT}} \text{lpr}[\text{SA}_0, E, P, A] \quad (3.3)$$

From (3.2) we have:

$$a^* \equiv \text{and}[\cdot A^*] \quad (3.4)$$

By (3.1) we see that  $P$  is a descendant of  $p$ . Hence we may apply the induction hypothesis to (3.3) and obtain:

$$\vdash_{\text{FOT}} \text{lpr}[\text{BSA}, E^*, Q, A^*]$$

for some  $Q$ . From (3.4) and (3.5) we can conclude:

$$\vdash_{\text{FOT}} \text{pr}[\text{BSA}, E^*, [a^*, \text{and\_I}[\cdot Q]], a^*] \quad (3.6)$$

We omit the proof of the remaining cases since they can be proved similarly as above. We have thus proved the first conjunct of our target proposition. We leave the proof of the remaining three conjuncts as an exercise for the reader. By these arguments we have shown that  $\text{SA}_0$  is a conservative extension of  $\text{BSA}$ .

We next show that  $\text{SA}$  is conservative over  $\text{SA}_0$ . Since the wffs in  $\text{SA}$  and  $\text{SA}_0$  are the same, we have only to show that:

$$\text{SA} \vdash a \implies \text{SA}_0 \vdash a$$

To show this it is sufficient to prove that:

$$\vdash_{\text{FOT}} \text{pr}[\text{SA}, E, p, a] \implies \vdash_{\text{FOT}} \text{pr}[\text{SA}_0, E, q, a] \text{ for some } q$$

In fact, as in the previous proof, we must prove a stronger proposition involving  $\text{lpr}$ ,  $\text{upr}$  and  $\text{eq\_pr}$  by the  $<$ -induction on  $p$ , but since the essential point lies in the proof of the above proposition (which is a conjunct of the stronger proposition) let us pretend that this is our target. So assume that:

$$\vdash_{\text{FOT}} \text{pr}[\text{SA}, E, p, a]$$



As before, we have 14 cases to consider, but the crucial case is (Case 1). Now (Case 1) produces 21 subcases depending on which axiom is actually used and the only nontrivial case is (Case 1.22) where (formal)  $<$ -induction is applied. In this case  $a$  is a wff in  $\mathbb{S}\mathbb{A}$  and we have for some  $p_0, w, z, a1, b, a0, t$ :

$$p \equiv [a, \text{axiom}[p_0]] \quad (1)$$

$$a \equiv \forall(z; \forall(w; w < z \rightarrow a1) \rightarrow a0) \rightarrow b. \quad (2)$$

$$\vdash_{\mathbb{S}\mathbb{A}} \text{pure\_variable}[z] \quad (3)$$

$$\vdash_{\mathbb{S}\mathbb{A}} \text{pure\_variable}[w] \quad (4)$$

$$\vdash_{\mathbb{S}\mathbb{A}} \text{ne}[z, w] \quad (5)$$

$$\vdash_{\mathbb{S}\mathbb{A}} \text{new}[w, a0] \quad (6)$$

$$\vdash_{\mathbb{S}\mathbb{A}} \text{wff}[a0] \quad (7)$$

$$\vdash_{\mathbb{S}\mathbb{A}} \text{term}[t] \quad (8)$$

$$\vdash_{\mathbb{S}\mathbb{A}} \text{substl}[w, z, a0, a1] \quad (9)$$

and

$$\vdash_{\mathbb{S}\mathbb{A}} \text{substl}[t, z, a0, b] \quad (10)$$

We show that  $a$  is provable in  $\mathbb{S}\mathbb{A}_0$  by using induction axiom in  $\mathbb{S}\mathbb{A}_0$ . Let us choose two distinct pure variables  $x$  and  $y$  which are new to  $a1$  and put:

$$A \equiv \forall(z; \forall(w; w < z \rightarrow a1) \rightarrow a0) \rightarrow \forall(w; w < z \rightarrow a1)$$

$$T \equiv [t. t]$$

$$B \equiv A_z[T]$$

$$A0 \equiv A_z[*]$$

$$IHx \equiv A_z[x]$$

$$IHy \equiv A_z[y]$$

$$A1 \equiv A_z[[x. y]]$$

$$A2 \equiv A_z[(x. y)]$$

Then we have:

$$\vdash_{\mathbb{S}\mathbb{A}_0} \text{axiom}[C] \quad (11)$$

where

$$C \equiv A0, \forall(x, y; IHx, IHy \rightarrow A1), \forall(x, y; IHx, IHy \rightarrow A2) \rightarrow B \quad (12)$$

Thus for some  $p_0$  we have:

$$\vdash_{\text{FOT}} \text{pr}[\mathbb{S}\mathbb{A}_0, [ ], [C, \text{axiom}[p_0]], C] \quad (13)$$

This means:

$$\mathbb{S}\mathbb{A}_0 \vdash C \quad (14)$$

By direct computation we have:

$$A0 \equiv \forall(z; \forall(w; w < z \rightarrow aI) \rightarrow a0) \rightarrow \forall(w; w < * \rightarrow aI) \quad (15)$$

Now to prove the wff  $a$  in  $\text{SA}_0$  let us work in  $\text{SA}_0$ . We first prove  $A0$ . Assume that:

$$\forall(z; \forall(w; w < z \rightarrow aI) \rightarrow a0) \quad (A1)$$

(In fact, this assumption will not be used.) Assume further that:

$$w < * \quad (A2)$$

Then we get  $aI$  by axiom 8 and  $(\perp E)$ . By discharging (A2) and then applying  $(\forall I)$  we get:

$$\forall(w; w < * \rightarrow aI)$$

We then get (15) by discharging (A1).

Next we prove:

$$\forall(x, y; IHx, IHy \rightarrow AI) \quad (16)$$

Assume  $IHx$ , that is:

$$\forall(z; \forall(w; w < z \rightarrow aI) \rightarrow a0) \rightarrow \forall(w; w < x \rightarrow aI) \quad (A3)$$

and  $IHy$ , that is:

$$\forall(z; \forall(w; w < z \rightarrow aI) \rightarrow a0) \rightarrow \forall(w; w < y \rightarrow aI) \quad (A4)$$

Now to prove  $AI$  we assume (A1) as before. Then we have:

$$\forall(w; w < x \rightarrow aI) \quad (C1)$$

and

$$\forall(w; w < y \rightarrow aI) \quad (C2)$$

By specializing  $z$  to  $x$  and  $y$  in (A1) we get

$$\forall(w; w < x \rightarrow aI) \rightarrow a0_z[x] \quad (C3)$$

and

$$\forall(w; w < y \rightarrow aI) \rightarrow a0_z[y] \quad (C4)$$

Using (C1)–(C4) we have:

$$a0_z[x] \quad (C5)$$

and

$$a0_z[y] \tag{C6}$$

Our goal is the conclusion of  $A1$ , that is:

$$\forall(w; w < [x.y] \rightarrow aI) \tag{G1}$$

To prove this assume:

$$w < [x.y] \tag{A5}$$

Then by axiom 14 we have 4 cases to consider. We will prove  $aI$  in all cases.

(Case 1) In this case we have  $w = x$ . Then by (C5) and ( $=subst$ ) we have  $a0_z[w]$ , that is,  $aI$ .

(Case 2) In this case we have  $w < x$ . By specializing (C1) to  $w$  we have  $aI$ .

(Case 3) and (Case 4) can be proved similarly.

From these we can deduce (G1). By discharging (A1) we get (16).

We can also prove:

$$\forall(x, y; IHx, IHy \rightarrow A2)$$

quite similarly.

Now in view of (12) we may conclude  $B$  that is:

$$\forall(z; \forall(w; w < z \rightarrow aI) \rightarrow a0) \rightarrow \forall(w; w < [t.t] \rightarrow aI) \tag{17}$$

On the other hand noting that  $t < [t.t]$  and  $b \equiv aI_w[t]$  we have:

$$\forall(w; w < [t.t] \rightarrow aI) \rightarrow b \tag{18}$$

By (17) and (18) we get  $a$ . We have thus proved the most crucial and the only nontrivial case.  $\square$

### 5.3. modification on FOT

The formal system FOT we introduced in Section 4.2 provided a general framework for defining arbitrary first order theories. We defined two first order theories, namely, BSA and SA in this framework and showed that these theories are equivalent. In the rest of the paper we will formally work in SA and will develop elementary mathematics in it. For this purpose, we find it better to modify FOT so that this enterprise will become easier to carry out. Roughly speaking, we will modify FOT so that all the theories defined by it will contain the language and axioms of SA. This means that in the modified FOT the empty list [ ] will define the axiom system SA. This modification prevents us from treating the systems not containing SA, however, for us this is not a

problem since we are not interested in such systems at the moment. On the other hand, the modified FOT can be used to define any extension of SA whose language and axioms are recursively enumerable.

We now modify FOT as follows. We replace the definitions of the predicates ‘term’, ‘wff’ and ‘axiom’ by the following three sexps *Term*, *Wff* and *Axiom*. We also define the predicate ‘sa\_axiom’ by the fourth sexp *Sa\_axiom*. From now on we will call thus obtained formal system as FOT.

```

+ term
  | x, S
    - variable[x]
  | *, S
  | [s . t], S
    - term[s, S]
    - term[t, S]
  | `(: /s ./t), S
    - term[s, S]
    - term[t, S]
  | (: apply . [Fun . T]), S
    - symbol[Fun]
    - term_list[T, S]
    - length[T, Arity]
    - theorem[function[[Fun . Arity]], S]
  | (: quote . t), S
  | (: * . t), S
  ;
+ wff
  | eq[s, t], S
    - term[s, S]
    - term[t, S]
  | lt[s, t], S
    - term[s, S]
    - term[t, S]
  | [Prd . T], S
    - admissible[Prd]
    - term_list[T, S]

```

```

    - length[T, Arity]
    - theorem[predicate[[Prd . Arity]], S]
| and [. A], S
    - wff_list[A, S]
| or [. A], S
    - wff_list[A, S]
| imply[A, b], S
    - wff_list[A, S]
    - wff[b, S]
| all[( : abs . [X, a]), S]
    - pure_variable_list[X]
    - wff[a, S]
| ex[( : abs . [X, a]), S]
    - pure_variable_list[X]
    - wff[a, S]
;
+ axiom
    | a, S
        - sa_axiom[a, S]
    | a, S
        - proves[p, axiom[a], S]
        - wff[a, S]
;
+ sa_axiom
    | '∀(s, t; [s . t] ≠ *), S % 1 %
    | '∀(s, t; (: s . t) ≠ *), S % 2 %
    | '∀(s, t, u, v; [s . t] ≠ (: u . v)), S % 3 %
    | '∀(s, t, u, v; [s . t] = [u . v] → s = u), S % 4 %
    | '∀(s, t, u, v; [s . t] = [u . v] → t = v), S % 5 %
    | '∀(s, t, u, v; (: s . t) = (: u . v) → s = u), S % 6 %
    | '∀(s, t, u, v; (: s . t) = (: u . v) → t = v), S % 7 %
    | '∀(r; ¬(r < *)), S % 8 %
    | '∀(r, s, t; ¬(r < (: s . t))), S % 9 %
    | '∀(s, t; s < [s, t]), S % 10 %
    | '∀(s, t; t < [s, t]), S % 11 %
    | '∀(r, s, t; r < s → r < [s, t]), S % 12 %

```

```

| '∀(r, s, t; r < t → r < [s.t]), S % 13 %
| '∀(r, s, t; r < [s.t] →
      r = s ∨ r < s ∨ r = t ∨ r < t), S % 14 %
| '((quote.*) = *), S % 15 %
| '((quote.[/s./t]) = [(quote./s).(quote./t)]), S % 16 %
| '((quote./[s./t]) = (: (quote./s).(quote./t))), S % 17 %
| '((*.*) = (: *. *)), S % 18 %
| '((*/s) = (: */u), (*./t) = (: */v) →
      (*.[/s./t]) = (: *.*./[u./v])), S % 19 %
| '((*/s) = (: */u), (*./t) = (: */v) →
      (*./[s./t]) = (: *.*.(/u./v))), S % 20 %
| '(/a0, ∀(/x, /y; /IHx, /IHx → /a1),
      ∀(/x, /y; /IHx, /IHx → /a2) → /b), S % 21 %
  - pure_variable[x]
  - pure_variable[y]
  - pure_variable[z]
  - ne[x, y]
  - wff[a, S]
  - new[x, a]
  - new[y, a]
  - term[t, S]
  - substl[t, z, a, b]
  - substl[* , z, a, a0]
  - substl[x, z, a, IHx]
  - substl[y, z, a, IHy]
  - substl[[x.y], z, a, a1]
  - substl['(: /x./y), z, a, a2]
| '∀(z; ∀(w; /w < /z → /a1) → /a) → /b), S % 22 %
  - pure_variable[z]
  - pure_variable[w]
  - ne[z, w]
  - wff[a, S]
  - new[w, a]
  - term[t, S]
  - substl[w, z, a, a1]
  - substl[t, z, a, b]

```

;

Let us refer to the old FOT as  $\text{FOT}_{old}$ . Then it is easy to see that:

$$\begin{aligned} & \vdash_{\text{FOT}_{old}} \text{pr}[\text{SA}, E, p, a] \text{ for some } p \\ & \iff \vdash_{\text{FOT}} \text{pr}[[ ], E, p, a] \text{ for some } q \end{aligned}$$

We therefore redefine SA as the empty list  $[ ]$ . The notation

$$S \vdash a$$

is an abbreviation of

$$\vdash_{\text{FOT}} \text{pr}[S, [ ], p, a] \text{ for some } p$$

where FOT now refers to the modified one.

## §6. Mathematics in SA

In this section we will formally work in the axiom system SA and will develop elementary mathematics within it.

### 6.1. elementary properties of SA

To begin with, let us prove some simple theorems in SA. We will display a formal theorem in SA in the following form.

**Thm 6.1.1.**  $s \leq t \leftrightarrow s < [t.t]$

*Proof.* Although this is a formal theorem, we give its proof informally. We prove the  $\leftarrow$  part first. Assume  $s < [t.t]$ . Then by axiom 14, we have

$$s = t \vee s < t \vee s = t \vee s < t$$

From this, by logic, we have  $s < t \vee s = t$  as desired. For the  $\rightarrow$  part, assume  $s \leq t$ , that is  $s < t \vee s = t$ . Then we have two cases.

(Case 1)  $s < t$ . In this case we have  $s < [t.t]$  by axiom 12.

(Case 2)  $s = t$ . By axiom 10, we have  $t < [t.t]$ . Since  $s = t$ , we have  $s < [t.t]$ .  $\square$

We give one more example that uses ordinary induction.

**Thm 6.1.2.**  $s < t, t < u \rightarrow s < u$

*Proof.* We prove this theorem by induction on  $u$ .

(Basis) Assume  $s < t$  and  $t < *$ . From the second assumption we can derive a contradiction by axiom 8.

(Step cons) We prove the theorem for  $[u1 . u2]$ , assuming the theorem for  $u1$  and  $u2$ . So assume  $s < t$  and  $t < [u1 . u2]$ . By the second assumption and axiom 14, we have four cases to consider.

(Case 1)  $t = u1$ . In this case we have  $s < u1$  by the first assumption. From this by axiom 12 we have  $s < [u1 . u2]$ .

(Case 2)  $t < u1$ . In this case we have  $s < u1$  by induction hypothesis. From this we have  $s < [u1 . u2]$  by axiom 12.

Cases 3 and 4 are proved similarly.

(Step snoc) We prove the theorem for  $(: u1 . u2)$ . Assume  $s < t$  and  $t < (: u1 . u2)$ . By the second assumption and axiom 9, we get a contradiction. From this we can deduce the desired result.  $\square$

## 6.2. abstract

In SA mathematical concepts are expressed in terms of wffs. For instance the concept of an atom is expressed in SA by the wff  $\exists(x, y; z = (: x . y))$  which means that  $z$  is an atom. As such mathematical concepts become more sophisticated the wff representing these concepts also become very complicated. We therefore need a systematic way of giving names to wffs in SA. We define the concept of an *abstract* for this purpose. Let  $x_1, \dots, x_n$  be a sequence of distinct pure variables and let  $d$  be a designator, i.e., wff or a term. Then the sexp:

$$\text{abstract}[[\text{abs} . [[x_1, \dots, x_n], d]]]$$

will be called an abstract of arity  $n$ . This sexp will also be written as:

$$\lambda(x_1, \dots, x_n; d)$$

Let  $A$  be an abstract of the above form and let  $t_1, \dots, t_n$  be a sequence of sexps. Then

$$A(t_1, \dots, t_n)$$

will denote the sexp:

$$d_{x_1, \dots, x_n} [[t_1, \dots, t_n]]$$

It is a wff.(term) if  $t_1, \dots, t_n$  are terms and  $d$  is a wff (term, resp.).

### Example 6.1.

$$\lambda(z; \exists(x, y; z = (: x . y)))(\text{apple}) \equiv \exists(x, y; \text{apple} = (: x . y)) \quad \square$$

An abstract  $A$  is called *closed* if  $A(*, \dots, *)$  is closed. A closed abstract will also



be called a *predicate* if it is abstracted from a wff. We will allow to give a name to a closed abstract. We will use strings of alphanumeric characters whose first characters are uppercase letters as names of abstracts. For instance, if we wish to assign a name 'Atom' to the abstract:

$$\lambda(z; \exists(x, y; z = (: x . y)))$$

we do it as follows:

Def 6.1.1.  $\text{Atom}(z) = \exists(x, y; z = (: x . y))$

Similarly we define an abstract (whose name is 'Mole') by:

Def 6.1.2.  $\text{Mole}(z) = \exists(x, y; z = [x . y])$

The general format of the definition of an abstract is as follows. Let  $d$  be a designator and let  $x_1, \dots, x_n$  be a sequence of distinct pure variables. Then:

$$\text{Def } n \text{ Name}(x_1, \dots, x_n) = d$$

gives the name *Name* to the abstract  $\lambda(x_1, \dots, x_n; d)$ . (This abstract must be closed.)

Let us define one more abstract.

Def 6.1.3.  $\text{Null}(z) = (z = *)$

Using these abstracts we can state some simple theorems whose proofs we do not give here.

Thm 6.2.1.  $\text{Null}(x) \vee \text{Mole}(x) \vee \text{Atom}(x)$

Thm 6.2.2.  $\text{Null}(x), \text{Mole}(x) \rightarrow \perp$

Thm 6.2.3.  $\text{Null}(x), \text{Atom}(x) \rightarrow \perp$

Thm 6.2.4.  $\text{Mole}(x), \text{Atom}(x) \rightarrow \perp$

These theorems are useful in proving the following theorems.

Thm 6.3.1.  $x = y \vee x \neq y$

Thm 6.3.2.  $x < y \vee \neg(x < y)$

We will also allow to give a name to a specific closed term in the following form:

Def 6.2.1.  $\text{FOT} = \text{'FOT}$

By this definition, FOT will denote the term 'FOT'. This definition also has the effect of making FOT a reserved string, thereby prohibiting the use of FOT as a variable. Let us give one more definition.

**Def 6.2.2.**  $SA = SA$

This definition simply makes SA a synonym for \*.

### 6.3. representation of Univ in SA

In order to formalize in SA what we have been doing informally in this paper, we must be able to formally define predicates whose informal counterparts have been given by inductive definitions. In view of Section 2, this will be accomplished if we can define the concept of a formal system in SA, or equivalently, if we can define the predicate *proves* in SA. By Theorem 2.1 we can reduce this problem to the problem of representing the single formal system Univ in SA. We can solve this problem by defining an abstract Univ\_tree such that Univ\_tree(T) means that T is a proof in the formal system Univ. (For technical reasons, proof trees represented by Univ\_tree are slightly different from the actual proof trees in Univ.)

Let us now define Univ\_tree. We first define Pnode as auxiliary concept.

**Def 6.3.1.**  $Pnode(t) = \exists(x, y; t = [(: x).y])$

Using Pnode, Univ\_tree is defined as follows.

**Def 6.3.2.**

$$\begin{aligned}
 & Univ\_tree(T) \\
 & = Pnode(T) \wedge \\
 & \forall(q \leq T; Pnode(q) \rightarrow \\
 & \quad \exists(u, v; q = [(: ne[*], [u.v])]) \\
 & \quad \vee \exists(u, v; q = [(: ne[*], (: u.v))]) \\
 & \quad \vee \exists(s, t; q = [(: ne[[s.t], *)])]) \\
 & \quad \vee \exists(s, t; q = [(: ne[(: s.t), *)])]) \\
 & \quad \vee \exists(s, t, u, v; q = [(: ne[[s.t], (: u.v))])]) \\
 & \quad \vee \exists(s, t, u, v; q = [(: ne[(: s.t), [u.v]])]) \\
 & \quad \vee \exists(s, t, u, v, Q; q = [(: ne[[s.t], [u.v]], [(: ne[s, u]).Q])]) \\
 & \quad \vee \exists(s, t, u, v, Q; q = [(: ne[[s.t], [u.v]], [(: ne[t, v]).Q])]) \\
 & \quad \vee \exists(s, t, u, v, Q; \\
 & \quad \quad q = [(: ne[(: s.t), (: u.v)], [(: ne[s, u]).Q])]) \\
 & \quad \vee \exists(s, t, u, v, Q; \\
 & \quad \quad q = [(: ne[(: s.t), (: u.v)], [(: ne[t, v]).Q])]) \\
 & \quad \vee \exists(x, v, L; q = [(: assoc[x, [[x.v].L], v)])]) \\
 & \quad \vee \exists(x, y, w, L, v, Q1, Q2;
 \end{aligned}$$

$$\begin{aligned}
& q = [(: \text{assoc}[x, [[y . w] . L], v]), \\
& \quad [(: \text{ne}[x, y]) . Q1], \\
& \quad [(: \text{assoc}[x, L, v]) . Q2]]) \\
\vee \exists(v, L; q = [(: \text{get}[* , [v . L], v])]) \\
\vee \exists(i, w, L, v, Q; \\
& \quad q = [(: \text{get}[[* . i], [w . L], v]), [(: \text{get}[i, L, v]) . Q]]) \\
\vee \exists(t, E, v, Q; q = [(: \text{eval}[( : \text{var} . t), E, v]), \\
& \quad [(: \text{assoc}[( : \text{var} . t), E, v]) . Q]]) \\
\vee \exists(E; q = [(: \text{eval} [* , E, *])]) \\
\vee \exists(s, t, E, u, v, Q1, Q2; \\
& \quad q = [(: \text{eval}[[s . t], E, [u . v]]), \\
& \quad [(: \text{eval}[s, E, u]) . Q1], \\
& \quad [(: \text{eval}[t, E, v]) . Q2]]) \\
\vee \exists(s, t, E, u, v, Q1, Q2; \\
& \quad q = [(: \text{eval}[(\text{snoc} . [s, t]), E, (: u . v)]), \\
& \quad [(: \text{eval}[s, E, u]) . Q1], \\
& \quad [(: \text{eval}[t, E, v]) . Q2]]) \\
\vee \exists(t, E; q = [(: \text{eval}[( : * . t), E, (: * . t)])]) \\
\vee \exists(t, E; q = [(: \text{eval}[( : \text{quote} . t), E, t])]) \\
\vee \exists(\text{Prd}, \text{FS}, \text{R}, i, c, \text{C}, \text{Env}, a, \text{A}, \text{P}, \text{Q1}, \text{Q2}, \text{Q3}, \text{Q4}, \text{Q5}; \\
& \quad q = [(: \text{proves}[[[\text{Prd}, i, \text{Env}] . \text{P}], [\text{Prd} . a], \text{FS}]), \\
& \quad [(: \text{assoc}[\text{Prd}, \text{FS}, \text{R}]) . Q1], \\
& \quad [(: \text{get}[i, \text{R}, [c . \text{C}])] . Q2), \\
& \quad [(: \text{eval}[c, \text{Env}, a]) . Q3], \\
& \quad [(: \text{eval}[\text{C}, \text{Env}, \text{A}]) . Q4], \\
& \quad [(: \text{lproves}[\text{P}, \text{A}, \text{FS}]) . Q5]]) \\
\vee \exists(\text{FS}; q = [(: \text{lproves}[[ ], [ ], \text{FS}])]) \\
\vee \exists(p, P, a, A, \text{FS}, \text{Q1}, \text{Q2}; \\
& \quad q = [(: \text{lproves}[[p . P], [a . A], \text{FS}]), \\
& \quad [(: \text{proves}[p, a, \text{FS}]) . Q1], \\
& \quad [(: \text{lproves}[P, A, \text{FS}]) . Q2]]) \\
\vee \exists(a, \text{FS}, p, Q; q = [(: \text{theorem}[a, \text{FS}]), \\
& \quad [(: \text{proves}[p, a, \text{FS}]) . Q]]) \\
)
\end{aligned}$$

Let us examine the basic structure of a `sexp` defined by `Univ_tree`. We first prove the following lemma.

**Thm 6.4.1.**  $\text{Univ\_tree}(T), \text{Pnode}(q), q < T \rightarrow \text{Univ\_tree}(q)$

*Proof.* Assume that:

$$\text{Univ\_tree}(T) \tag{1}$$

$$\text{Pnode}(q) \tag{2}$$

and

$$q < T \tag{3}$$

By the definition of  $\text{Univ\_tree}$  we have:

$$\text{Univ\_tree}(T) \equiv \text{Pnode}(T) \wedge \forall(q \leq T; \text{Pnode}(q) \rightarrow \mathcal{a})$$

where  $\mathcal{a}$  is a disjunction of 24 wffs. Hence by (1) we have:

$$\forall(q \leq T; \text{Pnode}(q) \rightarrow \mathcal{a}) \tag{4}$$

By (3) and (4) using the transitivity of  $<$  we have:

$$\forall(q \leq q; \text{Pnode}(q) \rightarrow \mathcal{a}) \tag{5}$$

(Recall that (5) is an abbreviation of  $\forall(q; q \leq \#q \rightarrow (\text{Pnode}(q) \rightarrow \mathcal{a}))$ .) From (2) and (5) we get  $\text{Univ\_tree}(q)$ .  $\square$

The lemma we just proved is useful in proving the following theorem which characterizes  $\text{Univ\_tree}$ . To state the theorem we introduce the following abstract.

**Def 6.4.1.**  $\text{Der}(T, c) = \text{Univ\_tree}(T) \wedge \exists(Q; T = [(: c). Q])$

**Thm 6.4.2.**

$$\begin{aligned} &\text{Univ\_tree}(T) \leftrightarrow \\ &\exists(u, v; T = [(: \text{ne}[*], [u.v])]) \\ &\vee \exists(u, v; T = [(: \text{ne}[*], (: u.v))]) \\ &\vee \exists(s, t; T = [(: \text{ne}[[s.t], *)])]) \\ &\vee \exists(s, t; T = [(: \text{ne}[(: s.t), *)])]) \\ &\vee \exists(s, t, u, v; T = [(: \text{ne}[[s.t], (: u.v)])]) \\ &\vee \exists(s, t, u, v; T = [(: \text{ne}[(: s.t), [u.v]])]) \\ &\vee \exists(s, t, u, v, T1; \text{Der}(T1, \text{ne}[s, u]) \wedge \\ &\quad T = [(: \text{ne}[[s.t], [u.v]), T1]) \\ &\vee \exists(s, t, u, v, T1; \text{Der}(T1, \text{ne}[t, v]) \wedge \\ &\quad T = [(: \text{ne}[[s.t], [u.v]), T1]) \\ &\vee \exists(s, t, u, v, T1; \text{Der}(T1, \text{ne}[s, u]) \wedge \\ &\quad T = [(: \text{ne}[(: s.t), (: u.v)], T1]) \\ &\vee \exists(s, t, u, v, T1; \text{Der}(T1, \text{ne}[t, v]) \wedge \end{aligned}$$

$$\begin{aligned}
& T = [(: \text{ne}[(: s . t), (: u . v)]), T1]) \\
\vee \exists(x, v, L; T = [(: \text{assoc}[x, [[x . v] . L], v)]) \\
\vee \exists(x, y, w, L, v, T1, T2; \\
& \quad \text{Der}(T1, \text{ne}[x, y]) \wedge \text{Der}(T2, \text{assoc}[x, L, v]) \\
& \quad \wedge T = [(: \text{assoc}[x, [[y . w] . L], v)], T1, T2]) \\
\vee \exists(v, L; T = [(: \text{get}[* , [v . L], v)]) \\
\vee \exists(i, w, L, v, T1; \text{Der}(T1, \text{get}[i, L, v]) \\
& \quad \wedge T = [(: \text{get}[[* . i], [w . L], v)], T1]) \\
\vee \exists(t, E, v, T1; \text{Der}(T1, \text{assoc}[(: \text{var} . t), E, v]) \\
& \quad \wedge T = [(: \text{eval}[(: \text{var} . t), E, v]), T1]) \\
\vee \exists(E; T = [(: \text{eval}[* , E, *)]) \\
\vee \exists(s, t, E, u, v, T1, T2; \\
& \quad \text{Der}(T1, \text{eval}[s, E, u]) \wedge \text{Der}(T2, \text{eval}[t, E, v]) \\
& \quad \wedge T = [(: \text{eval}[[s, t], E, [u . v]]), T1, T2]) \\
\vee \exists(s, t, E, u, v, T1, T2; \\
& \quad \text{Der}(T1, \text{eval}[s, E, u]) \wedge \text{Der}(T2, \text{eval}[t, E, v]) \\
& \quad \wedge T = [(: \text{eval}[(: \text{snoc} . [s, t]), E, (: u . v)]), T1, T2]) \\
\vee \exists(t, E; T = [(: \text{eval}[(: * . t), E, (: * . t)]) \\
\vee \exists(t, E; T = [(: \text{eval}[(: \text{quote} . t), E, t])]) \\
\vee \exists(\text{Prd}, \text{FS}, \text{R}, i, c, \text{C}, \text{Env}, a, \text{A}, \text{P}, T1, T2, T3, T4, T5; \\
& \quad \text{Der}(T1, \text{assoc}[\text{Prd}, \text{FS}, \text{R}]) \wedge \text{Der}(T2, \text{get}[i, \text{R}, [c . \text{C}]) \\
& \quad \wedge \text{Der}(T3, \text{eval}[c, \text{Env}, a]) \wedge \text{Der}(T4, \text{eval}[\text{C}, \text{Env}, \text{A}]) \\
& \quad \wedge \text{Der}(T5, \text{lproves}[\text{P}, \text{A}, \text{FS}]) \\
& \quad \wedge T = [(: \text{proves}[[[\text{Prd}, i, \text{Env}] . \text{P}], [\text{Prd} . a], \text{FS}]), \\
& \quad \quad T1, T2, T3, T4, T5]) \\
\vee \exists(\text{FS}; T = [(: \text{lproves}[[ ], [ ], \text{FS})]) \\
\vee \exists(p, P, a, \text{A}, \text{FS}, T1, T2; \\
& \quad \text{Der}(T1, \text{proves}[p, a, \text{FS}]) \wedge \text{Der}(T2, \text{lproves}[\text{P}, \text{A}, \text{FS}]) \\
& \quad \wedge T = [(: \text{lproves}[[p . P], [a . \text{A}], \text{FS}]), T1, T2]) \\
\vee \exists(a, \text{FS}, p, T1; \text{Der}(T1, \text{proves}[p, a, \text{FS}]) \\
& \quad \wedge T = [(: \text{theorem}[a, \text{FS}]), T1])
\end{aligned}$$

*Proof.* We first prove the  $\rightarrow$  part. Assume that  $\text{Univ\_tree}(T)$ . Then we have  $\text{Pnode}(T)$  and

$$\forall(q \preceq T; \text{Pnode}(q) \rightarrow a) \tag{1}$$

By specializing (1) to  $T$  and using the fact that  $\text{Pnode}(T)$ , we get  $b$  where  $b$  is

$a_q[[T]]$ . Since  $b$  is a disjunction of 24 wffs, we let  $b_i$  be the  $i$ -th disjunct of  $b$ . We now have 24 cases to consider where in the  $i$ -th case we may assume  $b_i$ . Since these cases may be treated rather similarly we only consider the case 17 as a typical case.

(Case 17) In this case we may assume  $b_{17}$ , that is:

$$\begin{aligned} \exists(s, t, E, u, v, Q1, Q2; \\ T = [(: \text{eval}[[s.t], E, [u.v]]), \\ [(: \text{eval}[s, E, u]).Q1], \\ [(: \text{eval}[t, E, v]).Q2]]) \end{aligned}$$

Let  $s, t, E, u, v, Q1$  and  $Q2$  be such that:

$$\begin{aligned} T = [(: \text{eval}[[s.t], E, [u.v]]), \\ [(: \text{eval}[s, E, u]).Q1], \\ [(: \text{eval}[t, E, v]).Q2]] \end{aligned} \tag{1}$$

We put

$$\begin{aligned} c &\equiv \text{eval}[[s.t], E, [u.v]], \\ T1 &\equiv [(: \text{eval}[s, E, u]).Q1] \end{aligned}$$

and

$$T2 \equiv [(: \text{eval}[t, E, v]).Q2]$$

Then by (1) we have:

$$T = [(: c), T1, T2]. \tag{2}$$

Now it is easy to see that  $\text{Pnode}(T1)$  and  $T1 < T$  hold. Then by Thm 6.4.1 we have:

$$\text{Univ\_tree}(T1) \tag{3}$$

Similarly we have:

$$\text{Univ\_tree}(T2) \tag{4}$$

From (3) and (4) we get:

$$\text{Der}(T1, \text{eval}[s, E, u]) \tag{5}$$

and

$$\text{Der}(T2, \text{eval}[t, E, v]) \tag{6}$$

Using (2), (5) and (6) we obtain:

$$\begin{aligned} \exists(s, t, E, u, v, T1, T2; \\ \text{Der}(T1, \text{eval}[s, E, u]) \wedge \text{Der}(T2, \text{eval}[t, E, v]) \\ \wedge T = [(: \text{eval}[[s.t], E, [u.v]]), T1, T2]) \end{aligned}$$

From this we obtain the desired result by ( $\vee I$ ).

We next prove the converse by the  $<$ -induction on  $T$ . We have 24 cases to consider but we only treat the following case:

(Case 17) In this case we can take  $s, t, E, u, v, T1$  and  $T2$  for which we have:

$$\text{Der}(T1, \text{eval}[s, E, u]) \tag{1}$$

$$\text{Der}(T2, \text{eval}[t, E, v]) \tag{2}$$

and

$$T = [(: \text{eval}[[s, t], E, [u.v]]), T1, T2] \tag{3}$$

By (3) we have:

$$\text{Pnode}(T) \tag{4}$$

Now take any  $q$  such that  $q \leq T$  and  $\text{Pnode}(q)$ . Then by (3), using the rules concerning  $<$ , we have:

$$q = T \vee q \leq T1 \vee q \leq T2$$

We have therefore three cases. First assume  $q = T$ . Then by (1), (2) and (3) we have:

$$\begin{aligned} \exists(s, t, E, u, v, Q1, Q2; \\ q = [(: \text{eval}[[s.t], E, [u.v]]), \\ [(: \text{eval}[s, E, u]). Q1], \\ [(: \text{eval}[t, E, v]). Q2]]) \end{aligned}$$

From this we have  $\omega$  by ( $\vee I$ ). Next assume  $q \leq T1$ . In this case we can easily derive  $\omega$  from (1). In case  $q \leq T2$ , we obtain  $\omega$  similarly by using (2). We therefore have:

$$\forall(q \leq T; \text{Pnode}(q) \rightarrow \omega) \tag{5}$$

From (4) and (5) we obtain  $\text{Univ\_tree}(T)$ .  $\square$

By Thm 6.4.2 we see that a  $\text{Univ\_tree}$  has a structure very similar to a proof in the formal system  $\text{U}_{\text{niv}}$ .

#### 6.4. formal systems in SA

Using the predicate `Univ_tree` we can now define the predicate `Proves` and related predicates as follows. By these predicates we can describe formal systems in SA.

**Def 6.5.1.**  $\text{Ne}(x, y) = \exists(T; \text{Der}(T, \text{ne}[x, y]))$

**Def 6.5.2.**  $\text{Assoc}(x, L, v) = \exists(T; \text{Der}(T, \text{assoc}[x, L, v]))$

**Def 6.5.3.**  $\text{Get}(i, L, v) = \exists(T; \text{Der}(T, \text{get}[i, L, v]))$

**Def 6.5.4.**  $\text{Eval}(t, \text{Env}, v) = \exists(T; \text{Der}(T, \text{eval}[t, \text{Env}, v]))$

**Def 6.5.5.**  $\text{Proves}(p, a, \text{FS}) = \exists(T; \text{Der}(T, \text{proves}[p, a, \text{FS}]))$

**Def 6.5.6.**  $\text{Lproves}(P, A, \text{FS}) = \exists(T; \text{Der}(T, \text{lproves}[P, A, \text{FS}]))$

**Def 6.5.7.**  $\text{Theorem}(a, \text{FS}) = \exists(T; \text{Der}(T, \text{theorem}[a, \text{FS}]))$

The following theorems show that these predicates have the desired properties.

**Thm 6.5.1.**  $\text{Ne}(x, y) \leftrightarrow x \neq y$

**Thm 6.5.2.**

$\text{Ne}(x, y) \leftrightarrow$

$\exists(u, v; x = * \wedge y = [u.v])$

$\vee \exists(u, v; x = * \wedge y = (: u.v))$

$\vee \exists(s, t; x = [s.t] \wedge y = *)$

$\vee \exists(s, t; x = (: s.t) \wedge y = *)$

$\vee \exists(s, t, u, v; x = [s.t] \wedge y = (: u.v))$

$\vee \exists(s, t, u, v; x = (: s.t) \wedge y = [u.v])$

$\vee \exists(s, t, u, v; x = [s.t] \wedge y = [u.v] \wedge \text{Ne}(s, u))$

$\vee \exists(s, t, u, v; x = [s.t] \wedge y = [u.v] \wedge \text{Ne}(t, v))$

$\vee \exists(s, t, u, v; x = (: s.t) \wedge y = (: u.v) \wedge \text{Ne}(s, u))$

$\vee \exists(s, t, u, v; x = (: s.t) \wedge y = (: u.v) \wedge \text{Ne}(t, v))$

**Thm 6.5.3.**

$\text{Assoc}(x, L, v) \leftrightarrow$

$\exists(L; \#L = [[x.v].L])$

$\vee \exists(y, w, L; \#L = [[y.w].L] \wedge \text{Ne}(x, y) \wedge \text{Assoc}(x, L, v))$

**Thm 6.5.4.**

$\text{Get}(i, L, v) \leftrightarrow$

$\exists(L; i = * \wedge \#L = [v.L])$



$$\vee \exists(i, w, L; \#i = [* . i] \wedge \#L = [w . L] \wedge \text{Get}(i, L, v))$$

Thm 6.5.5.

$$\begin{aligned} & \text{Eval}(t, \text{Env}, v) \leftrightarrow \\ & \exists(t; \#t = (: \text{var} . t) \wedge v = (: \text{var} . t)) \\ & \vee (t = * \wedge v = *) \\ & \vee \exists(s, t, u, v; \#t = [s . t] \wedge \#v = [u . v] \\ & \quad \wedge \text{Eval}(s, \text{Env}, u) \wedge \text{Eval}(t, \text{Env}, v)) \\ & \vee \exists(s, t, u, v; \#t = (: \text{snoc} . [s, t]) \wedge \#v = (: u . v) \\ & \quad \wedge \text{Eval}(s, \text{Env}, u) \wedge \text{Eval}(t, \text{Env}, v)) \\ & \vee \exists(t; \#t = (: * . t) \wedge v = (: * . t)) \\ & \vee \exists(t; \#t = (: \text{quote} . t) \wedge v = t) \end{aligned}$$

Thm 6.5.6.

$$\begin{aligned} & \text{Lproves}(P, A, \text{FS}) \leftrightarrow \\ & (P = [ ] \wedge A = [ ]) \\ & \vee \exists(p, a, P, A; \#P = [p . P] \wedge \#A = [a . A] \\ & \quad \wedge \text{Proves}(p, a, \text{FS}) \wedge \text{Lproves}(P, A, \text{FS})) \end{aligned}$$

Thm 6.5.7.

$$\begin{aligned} & \text{Proves}(p, a, \text{FS}) \leftrightarrow \\ & \exists(\text{Prd}, R, i, c, C, \text{Env}, a, A, P; \\ & \quad p = [[\text{Prd}, i, \text{Env}] . P] \wedge \#a = [\text{Prd} . a] \\ & \quad \wedge \text{Assoc}(\text{Prd}, \text{FS}, R) \wedge \text{Get}(i, R, [c . C]) \\ & \quad \wedge \text{Eval}(c, \text{Env}, a) \\ & \quad \wedge \text{Eval}(C, \text{Env}, A) \wedge \text{Lproves}(P, A, \text{FS})) \end{aligned}$$

Thm 6.5.8.  $\text{Theorem}(a, \text{FS}) \leftrightarrow \exists(p; \text{Proves}(p, a, \text{FS}))$

All of the above theorems can be proved without much difficulty. As an example let us sketch the proof of Thm 6.5.1.

*Proof of Thm 6.5.1.* We first prove the  $\rightarrow$  part. We prove the following by the  $<$ -induction on  $T$ .

$$\text{Der}(T, \text{ne}[x, y]) \rightarrow x \neq y$$

So assume  $\text{Der}(T, \text{ne}[x, y])$ , that is:

$$\text{Univ\_tree}(T) \wedge \exists(Q; T = [(: \text{ne}[x, y]) . Q])$$

Then applying Thm 6.4.2 we get  $b$  which is a disjunction of 24 wffs  $b_i$  ( $1 \leq i \leq 24$ ). Thus we have 24 cases and in (Case  $i$ ) we may assume  $b_i$ . But for  $i > 10$  we can

derive a contradiction by computing the *cbaar* of  $T$ . We therefore have only to consider (Case 1) – (Case 10).

(Case 1) Assume  $b_1$  that is  $\exists(u, v; T = [(: ne[*], [u.v])])$ . Let  $u$  and  $v$  be such that

$$T = [(: ne[*], [u.v])]$$

Then we have  $x = *$  and  $y = [u.v]$ . By axiom 1, we have  $x \neq y$ .

(Case 2) – (Case 6) can be proved similarly as (Case 1).

(Case 7) In this case we have:

$$\text{Der}(T1, ne[s, u]) \wedge T = [(: ne[[s.t], [u.v]), T1]$$

for some  $s, t, v$  and  $T1$ . Then we have  $x = [s.t]$  and  $y = [u.v]$ . Also we have  $T1 < T$  and since  $\text{Der}(T1, ne[s, u])$  we may apply induction hypothesis to conclude  $s \neq u$ . Then by axiom 4 we have  $[s.t] \neq [u.v]$ , that is,  $x \neq y$ .

(Case 8) – (Case 10) can be proved similarly as (Case 7).

Next we prove the  $\leftarrow$  part by induction on  $y$ .

(Basis) Assume  $x \neq *$ . Since  $\text{Null}(x) \vee \text{Mole}(x) \vee \text{Atom}(x)$  we have three cases. If  $\text{Null}(x)$  we have a contradiction. If  $\text{Mole}(x)$  we have  $s, t$  such that  $x = [s.t]$ . Then we can prove:

$$\text{Der}([: ne[[s.t, *]), ne[[s.t, *])$$

Hence we have  $\text{Ne}([s.t], *)$ , that is,  $\text{Ne}(x, *)$ . The case  $\text{Atom}(x)$  can be treated similarly.

(Step cons) We prove our target for  $[y1.y2]$  assuming it for  $y1$  and  $y2$ . Assume  $x \neq [y1.y2]$ . As before we can classify  $x$  into three cases. If  $x = *$ , we can prove  $\text{Ne}(x, [y1.y2])$  similarly as above. If  $\text{Atom}(x)$  then we have  $x = [x1.x2]$  for some  $x1$  and  $x2$ . By the decidability of  $=$ , we have  $x1 = y1 \vee x1 \neq y1$ . In either case we can prove easily that:

$$[x1.x2] \neq [y1.y2] \rightarrow x1 \neq y1 \vee x2 \neq y2$$

Now, in case  $x1 \neq y1$ , by induction hypothesis we have  $\text{Ne}(x1, x2)$ . We can then construct a  $T$ , for which we have

$$\text{Der}(T, ne[[x1.x2], [y1.y2]])$$

From this we have  $\text{Ne}(x, [y1.y2])$ .

(Step snoc) can be proved similarly.  $\square$

Using the predicates *Proves*, *Theorem* etc. we can formally talk about formal

systems. Let us give an example. Any closed term in SA denotes a unique sexp, and this relation can be defined by the formal system [*Denote*] where *Denote* is defined as follows:

+ denote  
 | \*, \*  
 | [s . t], [u . v]  
 - denote[s, u]  
 - denote[t, v]  
 | \(: /s . /t), (: u . v)  
 - denote[s, u]  
 - denote[t, v]  
 | (: quote . t), t  
 | (: \* . t), (: \* . t)  
 ;

For instance, the closed term 'x denotes the sexp x, since we have:

$$\vdash_{[Denote]} \text{denote}['x, x] \quad (1)$$

by the following reasoning. Let

$$p \equiv [[\text{denote}, [*, *, *], [[t . x]]]]$$

Then after some simple computations we have:

$$\text{proves}(p, \text{denote}['x, x], \text{Denote})$$

From this we have (1). We can formalize this by the following definitions:

Def 6.6.1.  $\text{Denote} = 'Denote$

Def 6.6.2.  $\text{Denote}(t, v) = \text{Theorem}(\text{denote}[t, v], \text{Denote})$

Then corresponding to (1), we can prove the following formally in SA.

$$\text{Theorem}(\text{denote}['x, 'x], \text{Denote}) \quad (2)$$

A formal proof of (2) can be obtained by translating the informal proof of (1).

Namely, in SA we can prove:

$$\text{Proves}('p, \text{denote}['x, 'x], \text{Denote})$$

and from this we have (2).

We can prove a more general theorem similarly. To state the theorem we introduce the following notation for any sexp x:

$$``x \text{ stands for } (: \text{quote} . x)$$

Thm 6.6.1.  $\text{Denote}('x, x)$

## §7. Metamathematics in SA

In this section we develop elementary metamathematics within SA by formalizing what we have done so far informally.

## 7.1. universality of FOT

We will prove that a formal version of Theorem 2.1 holds for FOT. In one direction, we have the following:

**Thm 7.1.1.**

$$\begin{aligned}
& (\text{Der}(T, \text{ne}[x, y]) \rightarrow \text{Theorem}(\text{ne}[x, y], \text{FOT})) \\
& \wedge (\text{Der}(T, \text{assoc}[x, L, v]) \rightarrow \text{Theorem}(\text{assoc}[x, L, v], \text{FOT})) \\
& \wedge (\text{Der}(T, \text{get}[i, L, v]) \rightarrow \text{Theorem}(\text{get}[i, L, v], \text{FOT})) \\
& \wedge (\text{Der}(T, \text{eval}[t, E, v]) \rightarrow \text{Theorem}(\text{eval}[t, E, v], \text{FOT})) \\
& \wedge (\text{Der}(T, \text{lproves}[P, A, \text{FS}]) \rightarrow \text{Theorem}(\text{lproves}[P, A, \text{FS}], \text{FOT})) \\
& \wedge (\text{Der}(T, \text{proves}[p, a, \text{FS}]) \rightarrow \text{Theorem}(\text{proves}[p, a, \text{FS}], \text{FOT})) \\
& \wedge (\text{Der}(T, \text{theorem}[a, \text{FS}]) \rightarrow \text{Theorem}(\text{theorem}[a, \text{FS}], \text{FOT}))
\end{aligned}$$

This theorem is proved by the  $<$ -induction on T. For the other direction, we have the following:

**Thm 7.1.2.**

$$\begin{aligned}
& (\text{Proves}(Q, \text{ne}[x, y], \text{FOT}) \rightarrow \text{Ne}(x, y)) \\
& \wedge (\text{Proves}(Q, \text{assoc}[x, L, v], \text{FOT}) \rightarrow \text{Assoc}(x, L, v)) \\
& \wedge (\text{Proves}(Q, \text{get}[i, L, v], \text{FOT}) \rightarrow \text{Get}(i, L, v)) \\
& \wedge (\text{Proves}(Q, \text{eval}[t, E, v], \text{FOT}) \rightarrow \text{Eval}(t, E, v)) \\
& \wedge (\text{Proves}(Q, \text{lproves}[P, A, \text{FS}], \text{FOT}) \rightarrow \text{Lproves}(P, A, \text{FS})) \\
& \wedge (\text{Proves}(Q, \text{proves}[p, a, \text{FS}], \text{FOT}) \rightarrow \text{Proves}(p, a, \text{FS})) \\
& \wedge (\text{Proves}(Q, \text{theorem}[a, \text{FS}], \text{FOT}) \rightarrow \text{Theorem}(a, \text{FS}))
\end{aligned}$$

This theorem can be proved by the  $<$ -induction on Q. Combining these theorems we have the following theorems.

**Thm 7.1.3.**  $\text{Ne}(x, y) \leftrightarrow \text{Theorem}(\text{ne}[x, y], \text{FOT})$

**Thm 7.1.4.**  $\text{Assoc}(x, L, v) \leftrightarrow \text{Theorem}(\text{assoc}[x, L, v], \text{FOT})$

**Thm 7.1.5.**  $\text{Get}(i, L, v) \leftrightarrow \text{Theorem}(\text{get}[i, L, v], \text{FOT})$

**Thm 7.1.6.**  $\text{Eval}(t, E, v) \leftrightarrow \text{Theorem}(\text{eval}[t, E, v], \text{FOT})$

**Thm 7.1.7.**  $\text{Lproves}(P, A, \text{FS}) \leftrightarrow \text{Theorem}(\text{lproves}[P, A, \text{FS}], \text{FOT})$

Thm 7.1.8.  $\text{Proves}(p, a, \text{FS}) \leftrightarrow \text{Theorem}(\text{proves}[p, a, \text{FS}], \text{FOT})$

Thm 7.1.9.  $\text{Theorem}(a, \text{FS}) \leftrightarrow \text{Theorem}(\text{theorem}[a, \text{FS}], \text{FOT})$

## 7.2. SA in SA

Let us study the axiom system SA within SA. First we give definitions concerning FOT.

Def 7.1.1.  $\text{Member}(x, L) = \text{Theorem}(\text{member}[x, L], \text{FOT})$

Def 7.1.2.  $\text{Non\_member}(x, L) = \text{Theorem}(\text{non\_member}[x, L], \text{FOT})$

Def 7.1.3.  $\text{Pure\_variable}(x) = \text{Theorem}(\text{pure\_variable}[x], \text{FOT})$

Def 7.1.4.  $\text{Pure\_variable\_list}(X) = \text{Theorem}(\text{pure\_variable\_list}[X], \text{FOT})$

Def 7.1.5.  $\text{Variable}(x) = \text{Theorem}(\text{variable}[x], \text{FOT})$

Def 7.1.6.  $\text{Pure\_part}(x, y) = \text{Theorem}(\text{pure\_part}[x, y], \text{FOT})$

Def 7.1.7.  $\text{Length}(X, n) = \text{Theorem}(\text{length}[X, n], \text{FOT})$

Def 7.1.8.  $\text{Symbol}(t) = \text{Theorem}(\text{symbol}[t], \text{FOT})$

Def 7.1.9.  $\text{Term}(t, S) = \text{Theorem}(\text{term}[t, S], \text{FOT})$

Def 7.1.10.  $\text{Term\_list}(T, S) = \text{Theorem}(\text{term\_list}[T, S], \text{FOT})$

Def 7.1.11.  $\text{Admissible}(p) = \text{Theorem}(\text{Admissible}[p], \text{FOT})$

Def 7.1.12.  $\text{Wff}(a, S) = \text{Theorem}(\text{wff}[a, S], \text{FOT})$

Def 7.1.13.  $\text{Wff\_list}(A, S) = \text{Theorem}(\text{wff\_list}[A, S], \text{FOT})$

Def 7.1.14.  $\text{Find}(t, x, y, v) = \text{Theorem}(\text{find}[t, x, y, v], \text{FOT})$

Def 7.1.15.  $\text{Rename}(x, y, z) = \text{Theorem}(\text{rename}[x, y, z], \text{FOT})$

Def 7.1.16.  $\text{Free}(t, X, v) = \text{Theorem}(\text{free}[t, X, v], \text{FOT})$

Def 7.1.17.  $\text{Subst}(T, X, a, b) = \text{Theorem}(\text{subst}[T, X, a, b], \text{FOT})$

Def 7.1.18.  $\text{Subst1}(t, x, a, b) = \text{Theorem}(\text{subst1}[t, x, a, b], \text{FOT})$

Def 7.1.19.  $\text{Bind\_find}(T, X, x, v) = \text{Theorem}(\text{bind\_find}[T, X, x, v], \text{FOT})$

Def 7.1.20.  $\text{Bind\_subst}(T, X, t, v) = \text{Theorem}(\text{bind\_subst}[T, X, t, v], \text{FOT})$

Def 7.1.21.  $\text{Append}(X, Y, Z) = \text{Theorem}(\text{append}[X, Y, Z], \text{FOT})$

Def 7.1.22.  $\text{Addend}(X, Y, Z) = \text{Theorem}(\text{addend}[X, Y, Z], \text{FOT})$

Def 7.1.23.  $\text{Vars}(a, X) = \text{Theorem}(\text{vars}[a, X], \text{FOT})$

Def 7.1.24.  $\text{Down}(X, U, V) = \text{Theorem}(\text{down}[X, U, V], \text{FOT})$

Def 7.1.25.  $\text{New}(x, t) = \text{Theorem}(\text{new}[x, t], \text{FOT})$

Def 7.1.26.  $\text{New\_list}(X, t) = \text{Theorem}(\text{new\_list}[X, t], \text{FOT})$

Def 7.1.27.  $\text{Eq\_pr}(S, E, P, U, V) = \text{Theorem}(\text{eq\_pr}[S, E, P, U, V], \text{FOT})$

Def 7.1.28.  $\text{Lpr}(S, E, P, A) = \text{Theorem}(\text{lpr}[S, E, P, A], \text{FOT})$

Def 7.1.29.  $\text{Upr}(S, A, E, P, c) = \text{Theorem}(\text{upr}[S, A, E, P, c], \text{FOT})$

Def 7.1.30.  $\text{Axiom}(a, S) = \text{Theorem}(\text{axiom}[a, S], \text{FOT})$

Def 7.1.31.  $\text{Sa\_axiom}(a, S) = \text{Theorem}(\text{sa\_axiom}[a, S], \text{FOT})$

Def 7.1.32.  $\text{Pr}(S, E, p, a) = \text{Theorem}(\text{pr}[S, E, p, a], \text{FOT})$

Def 7.1.33.  $\text{Thm}(a, S) = \exists(p; \text{Pr}(S, [ ], p, a))$

We can prove the following theorems that characterizes these predicates.

Thm 7.2.1.

$$\begin{aligned} \text{Member}(x, L) &\leftrightarrow \\ &\exists(X; L = [x.X]) \\ &\vee \exists(y, X; L = [y.X] \wedge \text{Member}(x, X)) \end{aligned}$$

Thm 7.2.2.

$$\begin{aligned} \text{Non\_member}(x, L) &\leftrightarrow \\ &(L = [ ]) \\ &\vee \exists(y, X; L = [y.X] \wedge \text{Ne}(x, y) \wedge \text{Non\_member}(x, X)) \end{aligned}$$

Thm 7.2.3.  $\text{Pure\_variable}(x) \leftrightarrow \exists(t; x = (: \text{var. } t))$

Thm 7.2.4.

$$\begin{aligned} \text{Pure\_variable\_list}(X) &\leftrightarrow \\ &(X = [ ]) \\ &\vee \exists(x, X; \#X = [x.X] \wedge \text{Pure\_variable}(x) \\ &\quad \wedge \text{Non\_member}(x, X) \wedge \text{Pure\_variable\_list}(X)) \end{aligned}$$

Thm 7.2.5.

$$\begin{aligned} \text{Variable}(x) &\leftrightarrow \\ &\text{Pure\_variable}(x) \\ &\vee \exists(x; \#x = (: \text{free. } x) \wedge \text{Variable}(x)) \end{aligned}$$

Thm 7.2.6.

$$\begin{aligned} \text{Pure\_part}(x, y) &\leftrightarrow \\ &\exists(t; x = (: \text{var. } t) \wedge y = (: \text{var. } t)) \\ &\vee \exists(x; \#x = (: \text{free. } x) \wedge \text{Pure\_part}(x, y)) \end{aligned}$$

Thm 7.2.7.

$$\begin{aligned} \text{Length}(X, n) &\leftrightarrow \\ &(X = * \wedge n = *) \\ &\vee \exists(X, n; \#X = [x.X] \wedge \#n = [* . n] \wedge \text{Length}(X, n)) \end{aligned}$$

Thm 7.2.8.  $\text{Symbol}(t) \leftrightarrow \exists(t; \#t = (: * . t))$

Thm 7.2.9.

$$\begin{aligned}
\text{Term}(t, S) \leftrightarrow & \\
& \text{Variable}(t) \\
& \vee (t = *) \\
& \vee \exists(s, t; \#t = [s . t]) \\
& \vee \exists(s, t; \#t = '( /s . /t)) \\
& \vee \exists(\text{Fun}, T; t = (: \text{apply} . [\text{Fun} . T]) \wedge \text{Symbol}(\text{Fun}) \\
& \quad \wedge \text{Term\_list}(T, S) \wedge \text{Length}(T, \text{Arity}) \\
& \quad \wedge \text{Theorem}(\text{function}[[\text{Fun} . \text{Arity}]], S)) \\
& \vee \exists(t; \#t = (: \text{quote} . t)) \\
& \vee \exists(t; \#t = (: * . t))
\end{aligned}$$

Thm 7.2.10.

$$\begin{aligned}
\text{Term\_list}(T, S) \leftrightarrow & \\
& (T = [ ]) \\
& \vee \exists(T; \#T = [t . T] \wedge \text{Term}(t, S) \wedge \text{Term\_list}(T, S))
\end{aligned}$$

Thm 7.2.11.

$$\begin{aligned}
\text{Admissible}(p) \leftrightarrow & \\
& \text{Symbol}(p) \wedge \text{Ne}(p, \text{and}) \wedge \text{Ne}(p, \text{or}) \wedge \text{Ne}(p, \text{imply}) \\
& \quad \wedge \text{Ne}(p, \text{all}) \wedge \text{Ne}(p, \text{ex})
\end{aligned}$$

Thm 7.2.12.

$$\begin{aligned}
\text{Wff}(a, S) \leftrightarrow & \\
& \exists(s, t; a = \text{eq}[s, t] \wedge \text{Term}(s, S) \wedge \text{Term}(t, S)) \\
& \vee \exists(s, t; a = \text{lt}[s, t] \wedge \text{Term}(s, S) \wedge \text{Term}(t, S)) \\
& \vee \exists(\text{Prd}, T; a = [\text{Prd} . T] \wedge \text{Admissible}(\text{Prd}) \\
& \quad \wedge \text{Term\_list}(T, S) \wedge \text{Length}(T, \text{Arity}) \\
& \quad \wedge \text{Theorem}(\text{predicate}[[\text{Prd} . \text{Arity}]], S)) \\
& \vee \exists(A; a = \text{and}[.A] \wedge \text{Wff\_list}(A, S)) \\
& \vee \exists(A; a = \text{or}[.A] \wedge \text{Wff\_list}(A, S)) \\
& \vee \exists(A, b; a = \text{imply}[A, b] \wedge \text{Wff\_list}(A, S) \wedge \text{Wff}(b, S)) \\
& \vee \exists(X, a; \#a = \text{all}[( : \text{abs} . [X, a])] \\
& \quad \wedge \text{Pure\_variable\_list}(X) \wedge \text{Wff}(a, S)) \\
& \vee \exists(X, a; \#a = \text{ex}[( : \text{abs} . [X, a])] \\
& \quad \wedge \text{Pure\_variable\_list}(X) \wedge \text{Wff}(a, S))
\end{aligned}$$

**Thm 7.2.13.**

$$\begin{aligned} \text{Wff\_list}(A, S) &\leftrightarrow \\ (A = [ ] & \\ \vee \exists(a, A; \#A = [a.A] \wedge \text{Wff}(a, S) \wedge \text{Wff\_list}(A, S)) & \end{aligned}$$

**Thm 7.2.14.**

$$\begin{aligned} \text{Find}(T, X, x, v) &\leftrightarrow \\ (T = [ ] \wedge X = [ ] \wedge v = x) & \\ \vee \exists(t, T, X; \#T = [t.T] \wedge \#X = [x.X] \wedge v = t) & \\ \vee \exists(t, T, y, X; \#T = [t.T] \wedge \#X = [y.X] & \\ \wedge \text{Ne}(x, y) \wedge \text{Find}(T, X, x, v)) & \end{aligned}$$

**Thm 7.2.15.**

$$\begin{aligned} \text{Rename}(x, y, z) &\leftrightarrow \\ (\text{Pure\_variable}(x) \wedge z = x) & \\ \vee (\text{Pure\_variable}(y) \wedge x = (: \text{free}.z)) & \\ \vee \exists(x, y, z; \#x = (: \text{free}.x) \wedge \#y = (: \text{free}.y) & \\ \wedge \#z = (: \text{free}.z) \wedge \text{Rename}(x, y, z)) & \end{aligned}$$

**Thm 7.2.16.**

$$\begin{aligned} \text{Free}(t, X, v) &\leftrightarrow \\ (t = * \wedge v = *) & \\ \vee \exists(x; v = (: \text{free}.t) \wedge \text{Variable}(t) \wedge \text{Pure\_part}(t, x) & \\ \wedge \text{Member}(x, X)) & \\ \vee \exists(x; v = t \wedge \text{Variable}(t) \wedge \text{Pure\_part}(t, x) & \\ \wedge \text{Non\_member}(x, X)) & \\ \vee \exists(s, t, u, v; \#t = [s.t] \wedge \#v = [u.v] & \\ \wedge \text{Free}(s, X, u) \wedge \text{Free}(t, X, v)) & \\ \vee \exists(s, t, u, v; \#t = `(: /s./t) \wedge \#v = `(: /u./v) & \\ \wedge \text{Free}(s, X, u) \wedge \text{Free}(t, X, v)) & \\ \vee \exists(t; \#t = (: *.t) \wedge v = (: *.t)) & \\ \vee \exists(t; \#t = (: \text{quote}.t) \wedge v = (: \text{quote}.t)) & \\ \vee \exists(f, T, V; t = (: \text{apply}.[f.T]) & \\ \wedge v = (: \text{apply}.[f.V]) \wedge \text{Free}(T, X, V)) & \end{aligned}$$

**Thm 7.2.17.**

$$\begin{aligned} \text{Subst}(T, X, a, b) &\leftrightarrow \\ (\text{Variable}(a) \wedge \text{Find}(T, X, a, b)) & \end{aligned}$$



$$\begin{aligned}
& \vee (a = * \wedge b = *) \\
& \vee \exists(s, t, u, v; a = [s.t] \wedge b = [u.v] \\
& \quad \wedge \text{Subst}(T, X, s, u) \wedge \text{Subst}(T, X, t, v)) \\
& \vee \exists(s, t, u, v; a = `(/s./t) \wedge b = `(/u./v) \\
& \quad \wedge \text{Subst}(T, X, s, u) \wedge \text{Subst}(T, X, t, v)) \\
& \vee \exists(t; a = (: *.t) \wedge b = (: *.t)) \\
& \vee \exists(t; a = (: quote.t) \wedge b = (: quote.t)) \\
& \vee \exists(f, U, V; a = (: apply.[f.U]) \\
& \quad \wedge b = (: apply.[f.V]) \wedge \text{Subst}(T, X, U, V)) \\
& \vee \exists(Y, t, v; a = (: abs.[Y,t]) \wedge b = (: abs.[Y,v]) \\
& \quad \wedge \text{Pure\_variable\_list}(Y) \wedge \text{Free}(T, Y, V) \\
& \quad \wedge \text{Free}(X, Y, Z) \wedge \text{Subst}(V, Z, t, v))
\end{aligned}$$

**Thm 7.2.18.**  $\text{Subst1}(t, x, s, v) \leftrightarrow \text{Subst}([t], [x], s, v)$

**Thm 7.2.19.**

$$\begin{aligned}
& \text{Bind\_find}(T, X, x, v) \leftrightarrow \\
& (T = [ ] \wedge X = [ ] \wedge x = v) \\
& \vee \exists(T, X; \#T = [v.T] \wedge \#X = [x.X]) \\
& \vee \exists(t, T, y, u; \#T = [t.T] \wedge \#X = [y.X] \wedge x \neq y \\
& \quad \wedge \text{Pure\_part}(x, u) \wedge \text{Pure\_part}(y, u) \wedge \text{Rename}(x, y, v)) \\
& \vee \exists(t, T, y, u1, u2; \#T = [t.T] \wedge \#X = [y.X] \wedge x \neq y \\
& \quad \wedge \text{Pure\_part}(x, u1) \wedge \text{Pure\_part}(y, u2) \wedge u1 \neq u2 \\
& \quad \wedge \text{Bind\_find}(T, X, x, v))
\end{aligned}$$

**Thm 7.2.20.**

$$\begin{aligned}
& \text{Bind\_subst}(T, X, t, v) \leftrightarrow \\
& (\text{Variable}(t) \wedge \text{Bind\_find}(T, X, t, v)) \\
& \vee (t = * \wedge v = *) \\
& \vee \exists(s, t, u, v; \#t = [s.t] \wedge \#v = [u.v] \\
& \quad \wedge \text{Bind\_subst}(T, X, s, u) \wedge \text{Bind\_subst}(T, X, t, v)) \\
& \vee \exists(s, t, u, v; \#t = (: s.t) \wedge \#v = (: u.v) \\
& \quad \wedge \text{Bind\_subst}(T, X, s, u) \wedge \text{Bind\_subst}(T, X, t, v)) \\
& \vee (t = v \wedge \text{Symbol}(t)) \\
& \vee \exists(t; \#t = (: quote.t) \wedge \#v = (: quote.t)) \\
& \vee \exists(f, U, V; t = (: apply.[f.U]) \wedge v = (: apply.[f.V]) \\
& \quad \wedge \text{Bind\_subst}(T, X, U, V)) \\
& \vee \exists(Y, t, v, V, Z; \#t = (: abs.[Y,t]) \wedge \#v = (: abs.[Y,v])
\end{aligned}$$

$$\begin{aligned} &\wedge \text{Pure\_variable\_list}(Y) \wedge \text{Free}(T, Y, V) \\ &\wedge \text{Free}(X, Y, Z) \wedge \text{Bind\_subst}(V, Z, t, v) \end{aligned}$$

**Thm 7.2.21.**

$$\begin{aligned} \text{Append}(X, Y, Z) &\leftrightarrow \\ & (X = [] \wedge Z = Y) \\ & \vee \exists(x, X, Z; \#X = [x.X] \wedge \#Z = [x.Z] \wedge \text{Append}(X, Y, Z)) \end{aligned}$$

**Thm 7.2.22.**

$$\begin{aligned} \text{Addend}(X, Y, Z) &\leftrightarrow \\ & (X = [] \wedge Z = Y) \\ & \vee \exists(x, X; \#X = [x.X] \wedge \text{Addend}(X, Y, Z) \wedge \text{Member}(x, Z)) \\ & \vee \exists(x, X, Z; \#X = [x.X] \wedge \#Z = [x.Z] \wedge \text{Addend}(X, Y, Z)) \end{aligned}$$

**Thm 7.2.23.**

$$\begin{aligned} \text{Vars}(t, V) &\leftrightarrow \\ & (\text{Variable}(t) \wedge V = [t]) \\ & \vee (t = * \wedge V = []) \\ & \vee \exists(s, t, S, T; \#t = [s.t] \wedge \text{Vars}(s, S) \wedge \text{Vars}(t, T) \\ & \quad \wedge \text{Addend}(S, T, V)) \\ & \vee \exists(s, t, S, T; \#t = (: /s. /t) \wedge \text{Vars}(s, S) \wedge \text{Vars}(t, T) \\ & \quad \wedge \text{Addend}(S, T, V)) \\ & \vee \exists(t; \#t = (: *. t) \wedge V = []) \\ & \vee \exists(t; \#t = (: \text{quote}. t) \wedge V = []) \\ & \vee \exists(f, T; t = (: \text{apply}. [f. T]) \wedge \text{Vars}(T, V)) \\ & \vee \exists(X, a, U; t = (: \text{abs}. [X, a]) \wedge \text{Vars}(a, U) \\ & \quad \wedge \text{Down}(X, U, V)) \end{aligned}$$

**Thm 7.2.24.**

$$\begin{aligned} \text{Down}(X, U, V) &\leftrightarrow \\ & (U = [] \wedge V = []) \\ & \vee \exists(x, U; \#U = [x.U] \wedge \text{Pure\_variable}(x) \\ & \quad \wedge \text{Member}(x, X) \wedge \text{Down}(X, U, V)) \\ & \vee \exists(x, U, V; \#U = [x.U] \wedge \#V = [x.V] \\ & \quad \wedge \text{Pure\_variable}(x) \wedge \text{Non\_member}(x, X) \\ & \quad \wedge \text{Down}(X, U, V)) \\ & \vee \exists(y, U, V; \#U = [(: \text{free}. y). U] \wedge \#V = [y.V] \\ & \quad \wedge \text{Pure\_part}(y, z) \wedge \text{Member}(z, X) \wedge \text{Down}(X, U, V)) \end{aligned}$$

$$\begin{aligned} & \vee \exists(y, U, V; \#U = [(: \text{free } y). U] \wedge \#V = [(: \text{free } y). V] \\ & \wedge \text{Pure\_part}(y, z) \wedge \text{Non\_member}(z, X) \\ & \wedge \text{Down}(X, U, V)) \end{aligned}$$

Thm 7.2.25.  $\text{New}(x, t) \leftrightarrow \exists(X; \text{Vars}(t, X) \wedge \text{Non\_member}(x, X))$

Thm 7.2.26.

$$\begin{aligned} & \text{New\_list}(X, t) \leftrightarrow \\ & (X = [ ]) \\ & \vee \exists(x, X; \#X = [x.X] \wedge \text{New}(x, t) \wedge \text{New\_list}(X, t)) \end{aligned}$$

Thm 7.2.27.

$$\begin{aligned} & \text{Eq\_pr}(S, E, P, U, V) \leftrightarrow \\ & (P = [ ] \wedge U = [ ] \wedge V = [ ]) \\ & \vee \exists(p, P, u, U, v, V; \#P = [p.P] \wedge \#U = [u.U] \\ & \wedge \#V = [v.V] \wedge \text{Pr}(S, E, p, u = v) \\ & \wedge \text{Eq\_pr}(S, E, P, U, V)) \end{aligned}$$

Thm 7.2.28.

$$\begin{aligned} & \text{Lpr}(S, E, P, A) \leftrightarrow \\ & (P = [ ] \wedge A = [ ]) \\ & \vee \exists(p, P, a, A; \#P = [p.P] \wedge \#A = [a.A] \\ & \wedge \text{Pr}(S, E, p, a) \wedge \text{Lpr}(S, E, P, A)) \end{aligned}$$

Thm 7.2.29.

$$\begin{aligned} & \text{Upr}(S, A, E, P, c) \leftrightarrow \\ & (A = [ ] \wedge P = [ ]) \\ & \vee \exists(a, A, p, P; \#A = [a.A] \wedge \#P = [p.P] \\ & \wedge \text{Pr}(S, [a.E], p, c) \wedge \text{Upr}(S, A, E, P, c)) \end{aligned}$$

Thm 7.2.30.

$$\begin{aligned} & \text{Axiom}(a, S) \leftrightarrow \\ & \text{Sa\_axiom}(a, S) \\ & \vee \exists(p; \text{Proves}(p, \text{axiom}[a], S) \wedge \text{Wff}(a, S)) \end{aligned}$$

Thm 7.2.31.

$$\begin{aligned} & \text{SA\_axiom}(a, S) \leftrightarrow \\ & (a = '\forall(s, t; [s.t] \neq *)') \\ & \vee (a = '\forall(s, t; (: s.t) \neq *)') \\ & \vee (a = '\forall(s, t, u, v; [s.t] \neq (: u.v))') \\ & \vee (a = '\forall(s, t, u, v; [s.t] = [u.v] \rightarrow s = u)') \end{aligned}$$

$$\begin{aligned}
& \vee (a = \forall(s, t, u, v; [s.t] = [u.v] \rightarrow t = v)) \\
& \vee (a = \forall(s, t, u, v; (: s.t) = (: u.v) \rightarrow s = u)) \\
& \vee (a = \forall(s, t, u, v; (: s.t) = (: u.v) \rightarrow t = v)) \\
& \vee (a = \forall(r; \neg(r < *))) \\
& \vee (a = \forall(r, s, t; \neg(r < (: s.t)))) \\
& \vee (a = \forall(s, t; s < [s.t])) \\
& \vee (a = \forall(s, t; t < [s.t])) \\
& \vee (a = \forall(r, s, t; r < s \rightarrow r < [s.t])) \\
& \vee (a = \forall(r, s, t; r < t \rightarrow r < [s.t])) \\
& \vee (a = \forall(r, s, t; r < [s.t] \rightarrow r = s \vee r < s \vee r = t \vee r < t)) \\
& \vee (a = \text{'(quote. *) = *}) \\
& \vee \exists(s, t; a = \text{'(quote. [s./t]) = [(quote./s).(quote./t)])}) \\
& \vee \exists(s, t; a = \text{'(quote./s./t) = (: (quote./s).(quote./t))}) \\
& \vee (a = \text{'(*.*) = (: *.*)}) \\
& \vee \exists(s, t, u, v; a = \text{'(*./s) = (: *.* /u), (*./t) = (: *.* /v) \rightarrow \\
& \quad (*.[s./t]) = (: *.* [u./v])}) \\
& \vee \exists(s, t, u, v; a = \text{'(*./s) = (: *.* /u), (*./t) = (: *.* /v) \rightarrow \\
& \quad (*./s./t) = (: *.* (/u./v))}) \\
& \vee \exists(a0, x, y, IHx, IHy, a1, a2, b, a; \\
& \quad \#a = \text{'(a0, \forall(x, /y; /IHx, /IHy \rightarrow /a1), \\
& \quad \quad \forall(x, /y; /IHx, /IHy \rightarrow /a2) \rightarrow /b)} \\
& \quad \wedge \text{Pure\_variable}(x) \wedge \text{Pure\_variable}(y) \\
& \quad \wedge \text{Pure\_variable}(z) \\
& \quad \wedge \text{Ne}(x, y) \wedge \text{New}(x, a) \wedge \text{New}(y, a) \wedge \text{Wff}(a, S) \\
& \quad \wedge \text{Term}(t, S) \\
& \quad \wedge \text{Subst } 1(t, z, a, b) \wedge \text{Subst } 1(*, z, a, IHx) \\
& \quad \wedge \text{Subst } 1(y, z, a, IHy) \\
& \quad \wedge \text{Subst } 1([x.y], z, a, a1) \wedge \text{Subst } 1(\text{'(: /x./y)}, z, a, a2)) \\
& \vee \exists(z, w, a1, a, b; \\
& \quad \#a = \text{'(\forall(z; \forall(w; /w < /z \rightarrow /a1) \rightarrow /a) \rightarrow /b)} \\
& \quad \wedge \text{Pure\_variable}(z) \wedge \text{Pure\_variable}(w) \\
& \quad \wedge \text{Ne}(z, w) \wedge \text{New}(w, a) \\
& \quad \wedge \text{Wff}(a, S) \wedge \text{Term}(t, S) \wedge \text{Subst } 1(w, z, a, a1) \\
& \quad \wedge \text{Subst } 1(t, z, a, b))
\end{aligned}$$

**Thm 7.2.32.**

$$\begin{aligned}
& \text{Pr}(S, E, p, a) \leftrightarrow \\
& (p = [a, \text{axiom}[]] \wedge \text{Axiom}(a, S) \wedge \text{Wff\_list}(E, S)) \\
& \vee (p = [a, \text{assumption}[]] \wedge \text{Member}(a, E) \wedge \text{Wff\_list}(E, S)) \\
& \vee \exists(A, P; p = [\text{and}[\cdot A], \text{and\_I}[\cdot P]] \wedge a = \text{and}[\cdot A] \\
& \quad \wedge \text{Lpr}(S, E, P, A)) \\
& \vee \exists(p, A; \#p = [a, \text{and\_E}[p]] \wedge \text{Pr}(S, E, p, \text{and}[\cdot A]) \\
& \quad \wedge \text{Member}(a, A)) \\
& \vee \exists(A, p; \#p = [\text{or}[\cdot A], \text{or\_I}[p]] \wedge a = \text{or}[\cdot A] \\
& \quad \wedge \text{Pr}(S, E, p, a) \wedge \text{Member}(a, A) \wedge \text{Wff\_list}(A, S)) \\
& \vee \exists(p, P, A; \#p = [a, \text{or\_E}[p, P]] \wedge \text{Pr}(S, E, p, \text{or}[\cdot A]) \\
& \quad \wedge \text{Upr}(S, A, E, P, a)) \\
& \vee \exists(A, b, p, F; \#p = [\text{imply}[A, b], \text{imply\_I}[p]] \\
& \quad \wedge a = \text{imply}[A, b] \wedge \text{Append}(A, E, F) \wedge \text{Pr}(S, F, p, b)) \\
& \vee \exists(p, P, A; \#p = [a, \text{imply\_E}[p, P]] \\
& \quad \wedge \text{Pr}(S, E, p, \text{imply}[A, a]) \wedge \text{Lpr}(S, E, P, A)) \\
& \vee \exists(X, a, Y, p, b; \#p = [\text{all}[(\cdot \text{abs} \cdot [X, a]), \text{all\_I}[[X, Y], p]] \\
& \quad \wedge \#a = \text{all}[(\cdot \text{abs} \cdot [X, a])] \\
& \quad \wedge \text{New\_list}(Y, E) \wedge \text{Pure\_variable\_list}(X) \wedge \text{Wff}(a, S) \\
& \quad \wedge \text{Subst}(Y, X, a, b) \wedge \text{Pr}(S, E, p, b)) \\
& \vee \exists(T, p, X, b; \#p = [a, \text{all\_E}[T, p]] \\
& \quad \wedge \text{Pr}(S, E, p, \text{all}[(\cdot \text{abs} \cdot [X, b])]) \\
& \quad \wedge \text{Term\_list}(T, S) \wedge \text{Subst}(T, X, b, a)) \\
& \vee \exists(X, b, T, p; \#p = [\text{ex}[(\cdot \text{abs} \cdot [X, b]), \text{ex\_I}[T, p]] \\
& \quad \wedge a = \text{ex}[(\cdot \text{abs} \cdot [X, b])] \wedge \text{Term\_list}(T, S) \\
& \quad \wedge \text{Pure\_variable\_list}(X) \wedge \text{Wff}(b, S) \\
& \quad \wedge \text{Subst}(T, X, b, a) \wedge \text{Pr}(S, E, p, a)) \\
& \vee \exists(Y, p, q, b, c; \#p = [a, \text{ex\_E}[Y, p, q]] \wedge \text{New\_list}[Y, E] \\
& \quad \wedge \text{New\_list}(Y, a) \wedge \text{Pr}(S, E, p, \text{ex}[(\cdot \text{abs} \cdot [X, b])]) \\
& \quad \wedge \text{Subst}(Y, X, b, c) \wedge \text{Pr}(S, [c \cdot E], q, a)) \\
& \vee \exists(t; p = [t = t, \text{axiom\_id}[]] \wedge \text{Term}(t, S)) \\
& \vee \exists(U, V, X, c, p, P, b; \#p = [a, \text{subst}[[U, V, X, c], p, P]] \\
& \quad \wedge \text{Wff}(c, S) \wedge \text{Subst}(U, X, c, b) \wedge \text{Subst}(V, X, c, a) \\
& \quad \wedge \text{Pr}(S, E, p, b) \wedge \text{Eq\_pr}(S, E, P, U, V))
\end{aligned}$$

Using these theorems we can study about SA and its extensions within

SA. For example corresponding to the simple metatheorem:

$$S \vdash a \rightarrow b, S \vdash a \implies S \vdash b$$

we have the following theorem:

**Thm 7.3.1.**  $\text{Thm}(a \rightarrow b, S), \text{Thm}(a, S) \rightarrow \text{Thm}(b, S)$

*Proof.* Assume  $\text{Thm}(a \rightarrow b, S)$  and  $\text{Thm}(a, S)$ . Then we have  $p$  and  $q$  such that:

$$\text{Pr}(S, [ ], p, a \rightarrow b) \quad (1)$$

$$\text{Pr}(S, [ ], q, a) \quad (2)$$

From (2) and Thm 7.2.28 we have:

$$\text{Lpr}(S, [ ], [q], [a]) \quad (3)$$

By (1), (3) and Thm 7.2.32 we have:

$$\text{Pr}(S, [ ], [b, \text{imply\_E}[p, [q]]], b) \quad (4)$$

Then by applying  $(\exists I)$  to (4) we have  $\text{Thm}(b, S)$ .  $\square$

We can similarly prove the following theorems and much more similar theorems which we do not list here.

**Thm 7.3.2.**  $\text{Thm}(a, SA) \rightarrow \text{Thm}(a, S)$

**Thm 7.3.3.**  $\text{Wff}(a, S) \rightarrow \text{Thm}(\perp \rightarrow a, S)$

**Thm 7.3.4.**  $\text{Thm}(\ulcorner t = \urcorner t, SA)$

**Thm 7.3.5.**  $\text{Thm}(\ulcorner [s.t] = \urcorner [s.\urcorner t], SA)$

**Thm 7.3.6.**  $\text{Thm}(\ulcorner (\ulcorner : s.t) = (\ulcorner : \urcorner s.\urcorner t) \urcorner, SA)$

Next we will prove that SA is *inductively complete* in the sense that if Theorem  $(a, FS)$  then we can formally prove it in SA. (We borrowed this terminology from Feferman [4].) We can state this formally as follows:

$$\text{Theorem}(a, FS) \rightarrow \text{Thm}(\ulcorner \text{Theorem}(\ulcorner a, \urcorner FS) \urcorner, SA)$$

We first prove the following key lemma.

**Thm 7.3.7.**  $\text{Univ\_tree}(T) \rightarrow \text{Thm}(\ulcorner \text{Univ\_tree}(\ulcorner T) \urcorner, SA)$

*Proof.* We prove this theorem by the  $<$ -induction on  $T$ . Assume  $\text{Univ\_tree}(T)$ . Then by Thm 6.4.2 we have 24 cases to consider of which we treat only (Case 17) as a typical case.

(Case 17) In this case we have some  $s, t, E, u, v, T1$  and  $T2$  such that:

$$\text{Der}(T1, \text{eval}[s, E, u]) \quad (1)$$

$$\text{Der}(T2, \text{eval}[t, E, v]) \quad (2)$$

and

$$T = [(: \text{eval}[s . t], E, [u . v]), T1, T2] \quad (3)$$

From (1) we have:

$$\text{Univ\_tree}(T) \quad (4)$$

and

$$\exists(Q; T1 = [(: \text{eval}[s, E, u]) . Q]) \quad (5)$$

By (3) we have  $T1 < T$  and we can apply induction hypothesis to (4) and get:

$$\text{Thm}(\text{Univ\_tree}(/^{\wedge}T1), SA) \quad (6)$$

Now by (5) we can take  $Q$  such that:

$$T1 = [(: \text{eval}[s, E, u]) . Q] \quad (7)$$

By Thm 7.3.4 we have:

$$\text{Thm}(\text{``}T1 = \text{``}T1, SA) \quad (8)$$

We can rewrite (8) using (7), Thm 7.3.5 and Thm 7.3.6 as follows:

$$\text{Thm}(\text{``}T1 = [(: \text{eval}[/^{\wedge}s, /^{\wedge}E, /^{\wedge}u]) . /^{\wedge}Q], SA) \quad (9)$$

From (6) and (9) using Thm 7.2.32 we get:

$$\text{Thm}(\text{Der}(/^{\wedge}T1, \text{eval}[/^{\wedge}s, /^{\wedge}E, /^{\wedge}u]), SA) \quad (10)$$

From (2) we have the following similarly:

$$\text{Thm}(\text{Der}(/^{\wedge}T2, \text{eval}[/^{\wedge}t, /^{\wedge}E, /^{\wedge}v]), SA) \quad (11)$$

We can also prove the following by using (3):

$$\text{Thm}(\text{``}T = [(: \text{eval}[/^{\wedge}s . /^{\wedge}t], /^{\wedge}E, [ /^{\wedge}u . /^{\wedge}v]), /^{\wedge}T1, /^{\wedge}T2], SA) \quad (12)$$

From (10), (11) and (12) we can prove:

$$\begin{aligned} &\text{Thm}(\exists(s, t, E, u, v, T1, T2; \\ &\text{Der}(T1, \text{eval}[s, E, u]) \wedge \text{Der}(T2, \text{Eval}[t, E, v]) \\ &\wedge T = [(: \text{eval}[[s . t], E, [u . v]), T1, T2]), SA) \end{aligned} \quad (13)$$

Now let  $b$  be the right hand side of the equivalence of Thm 6.4.2. Then applying  $(\vee I)$  to (13) we have:

$$\text{Thm}(\mathbf{b}_T[\wedge T], \text{SA}) \quad (14)$$

On the other hand, repeating the proof of Thm 6.4.2 within SA, we obtain:

$$\text{Thm}(\text{Univ\_tree}(\wedge T) \leftrightarrow \mathbf{b}_T(\wedge T), \text{SA}) \quad (15)$$

By (14) and (15) we have:

$$\text{Thm}(\text{Univ\_tree}(\wedge T)), \text{SA})$$

□

Using Thm 7.3.7 we can easily prove the inductive completeness of SA:

**Thm 7.3.8.**  $\text{Theorem}(a, \text{FS}) \rightarrow \text{Thm}(\text{Theorem}(\wedge a, \wedge \text{FS}), \text{SA})$

We also have the following theorem as a corollary to this theorem.

**Thm 7.3.9.**  $\text{Thm}(a, \text{S}) \rightarrow \text{Thm}(\text{Thm}(\wedge a, \wedge \text{S}), \text{SA})$

## §8. Incompleteness Theorem

In this section we prove some of Gödel's incompleteness theorems (Gödel[5]) including the second incompleteness theorem formally in SA.

### 8.1. reflection principle

Let us make some observations about what we are doing by looking at it from *outside*. We have been developing our informal theory of symbolic expressions only using constructively acceptable arguments. We also claim that our formal theory SA reflects faithfully part of our informal mathematics. This means firstly that each wff in SA can be translated into an informal statement that is meaningful to our informal mathematics, and secondly that each formal proof of a wff in SA can be translated into an acceptable informal proof of the corresponding informal statement. We may call this translation process as *informalization*. Since the translation process should be almost clear, we only explain how terms in SA are translated into informal expressions denoting sexps. We denote the translation of a term  $t$  by  $\tilde{t}$ . If  $t$  is a variable then we translate it into an informal variable. E.g., if  $t$  is 'x' then we translate it into 'x'. We translate  $*$  into  $*$ . The translation of  $[s . t]$  is  $[\tilde{s} . \tilde{t}]$  and the translation of  $(: s . t)$  is  $(\tilde{s} . \tilde{t})$ . The translation of ' $t$  is  $t$ ' and the translation of  $(* . t)$  is  $(* . \tilde{t})$ .

Let us consider the translation of formal developments in Section 6 and 7. The corresponding informal developments considerably overlaps with our earlier developments in Sections 1–5. But there are minor differences which we now



explain. In Section 2 we defined the concept of a formal system using informal inductive definitions. Namely, we defined the informal predicates *ne*, *assoc*, *get*, *eval*, *lproves*, *proves* and *theorem* by informal inductive definitions. On the other hand, by translating Def 6.1.1–7 in Section 6.4 we have the explicit definitions of the informal predicates *Ne*, *Assoc*, *Get*, *Eval*, *Lproves*, *Proves* and *Theorem*. It is, however, easy to see informally that these two groups of concepts are equivalent. From this observation, it follows, for instance, that the concept defined by the translation of Def 7.1.32 in Section 7.2 is equivalent to the notion of the provability in a formal theory which we defined in Section 4.2.

We can thus conclude that each formal theorem of SA yields as by-product an informal theorem which is its informal counterpart. Such an informal theorem may sometimes be used to produce another formal theorem. We give two important examples of this.

By informalizing Thm 7.3.8 we have the following theorem:

**Theorem 8.1.**  $\vdash_{FS} a \Rightarrow SA \vdash \text{Theorem}(a, 'FS)$

The logic programming language Qute [12] can be used to verify  $\vdash_{FS} a$  automatically, so that this theorem will be useful when we implement a proof checking system for SA on a computer.

The following theorem can be obtained by reading Thm 7.3.9 informally.

**Theorem 8.2.**  $S \vdash a \Rightarrow SA \vdash \text{Thm}(a, 'S)$

This theorem is the first Löb derivability condition and Thm 7.3.9 is the second. We have already proved the third derivability condition as Thm 7.3.1. (See, e.g., Feferman [4] for these derivability conditions.)

## 8.2. diagonalization lemma

We prove the diagonalization lemma in this subsection. To state the diagonalization lemma, we need some definitions.

**Def 8.1.1.**  $\text{Sub}(t, a, b) = \exists(x; \text{Vars}(a, [x]) \wedge \text{Subst1}('t, x, a, b))$

**Def 8.1.2.**  $A(x, S) = \exists(X; \text{Sub}(x, x, X) \wedge \neg \text{Thm}(X, S))$

**Def 8.1.3.**  $B(S) = 'A(x, /'S)$

**Def 8.1.4.**  $C(S) = A(B(S), S)$

**Def 8.1.5.**  $D(S) = 'A(/'B(S), /'S)$

Using these predicates, we can state the diagonalization lemma as follows.

$$\text{Thm}(D(S) \leftrightarrow \neg \text{Thm}(/^{\wedge}D(S), /^{\wedge}S), SA)$$

To prove this we prepare some auxiliary lemmas. We can easily prove the following two lemmas:

$$\text{Thm 8.1.1. } \text{Sub}(t, a, b1), \text{Sub}(t, a, b2) \rightarrow b1 = b2$$

$$\text{Thm 8.1.2. } \text{Sub}(B(S), B(S), D(S))$$

Then we can prove the following lemma.

$$\text{Thm 8.1.3. } C(S) \leftrightarrow \neg \text{Thm}(D(S), S)$$

*Proof.* We first prove the  $\rightarrow$  part. Assume  $C(S)$  and let  $X$  be such that:

$$\text{Sub}(B(S), B(S), X) \wedge \neg \text{Thm}(X, S)$$

This implies:

$$\text{Sub}(B(S), B(S), X) \tag{1}$$

and

$$\neg \text{Thm}(X, S) \tag{2}$$

Then by (1), Thm 8.1.2 and Thm 8.1.1 we have:

$$X = D(S) \tag{3}$$

By (2) and (3) we have  $\neg \text{Thm}(D(S), S)$ .

Next, we prove the  $\leftarrow$  part. Assume  $\text{Thm}(D(S), S)$ . Then by Thm 8.1.2 we have:

$$\text{Sub}(B(S), B(S), D(S)) \wedge \neg \text{Thm}(D(S), S)$$

From this we have  $C(S)$  by  $(\exists I)$ .  $\square$

We have the following by applying  $(\forall I)$  to Thm 8.1.3.

$$SA \vdash \forall(S; C(S) \leftrightarrow \neg \text{Thm}(D(S), S))$$

From this by Theorem 8.2 we have the following:

$$SA \vdash \text{Thm}(\forall(S; C(S) \leftrightarrow \neg \text{Thm}(D(S), S)), SA)$$

We therefore have the following theorem.

$$\text{Thm 8.1.4. } \text{Thm}(\forall(S; C(S) \leftrightarrow \neg \text{Thm}(D(S), S)), SA)$$

We can now prove the diagonalization lemma as follows.

$$\text{Thm 8.1.5. } \text{Thm}(D(S) \leftrightarrow \neg \text{Thm}(/^{\wedge}D(S), /^{\wedge}S), SA)$$

*Proof.* By a simple but tedious computation we have:

$$\forall(S; C(S) \leftrightarrow \neg \text{Thm}(D(S), S)) = \forall(S; C(S) \leftrightarrow \neg \text{Thm}(D(S), S)) \quad (1)$$

By (1) and Thm 8.1.4 we have:

$$\text{Thm}(\forall(S; C(S) \leftrightarrow \neg \text{Thm}(D(S), S)), SA) \quad (2)$$

By formally specializing 'S' to the term 'S we obtain:

$$\text{Thm}(C(/S) \leftrightarrow \neg \text{Thm}(D(/S), /S)), SA) \quad (3)$$

On the other hand we have the following by a simple calculation:

$$\text{Thm}(D(/S) = D(S), SA) \quad (4)$$

By (3) and (4) we have the following as desired:

$$\text{Thm}(D(S) \leftrightarrow \neg \text{Thm}(D(S), /S), SA)$$

□

*Remark.* The method we used to obtain Thm 8.1.5 from Thm 8.1.3 is general and applicable to similar cases. We will call this method as *formalization*. Thus, e.g., we say that Thm 8.1.5 is obtained by formalizing Thm 8.1.3. □

### §.3. incompleteness theorems

We first define the concept of *consistency*.

Def §.2.1.  $\text{Consis}(S) = \neg \text{Thm}(\perp, S)$

$\text{Consis}(S)$  says that S is consistent.

We will prove the following theorems.

Thm §.2.1.  $\text{Thm}(D(S), S) \rightarrow \text{Thm}(\perp, S)$

Thm §.2.2.  $\text{Consis}(S) \rightarrow \neg \text{Thm}(D(S), S)$

Thm §.2.3.  $\text{Consis}(S) \rightarrow C(S)$

Thm §.2.4.  $\text{Thm}(\text{Consis}(/S) \rightarrow D(S), SA)$

Thm §.2.5.  $\text{Thm}(\text{Consis}(/S) \rightarrow D(S), S)$

Thm §.2.6.  $\text{Thm}(\text{Consis}(/S), S) \rightarrow \text{Thm}(D(S), S)$

Thm §.2.7.  $\text{Consis}(S) \rightarrow \neg \text{Thm}(\text{Consis}(/S), S)$

*Proof of Thm 8.2.1.* Assume:

$$\text{Thm}(D(S), S) \quad (1)$$

By Thm 7.3.9 and (1) we have:

$$\text{Thm}(\text{Thm}(\wedge D(S), \wedge S), SA) \quad (2)$$

By (2) and Thm 7.3.2 we have:

$$\text{Thm}(\text{Thm}(\wedge D(S), \wedge S), S) \quad (3)$$

On the other hand by Thm 8.1.5 and Thm 7.3.2 we have:

$$\text{Thm}(D(S) \rightarrow \neg \text{Thm}(\wedge D(S), \wedge S), S) \quad (4)$$

By (1), (4) and Thm 7.3.1 we have:

$$\text{Thm}(\neg \text{Thm}(\wedge D(S), \wedge S), S) \quad (5)$$

By (3), (5) and Thm 7.3.1 we have  $\text{Thm}(\perp, S)$ .  $\square$

Thm 8.2.2 is a logical consequence of Thm 8.2.1. Thm 8.2.2 says that if  $S$  is consistent then the ‘formula’  $D(S)$  which states its own unprovability is in fact unprovable. Here we note that  $S$  must be an extension of  $SA$  according to our definition of FOT, and that Thm 7.3.2 states this fact formally.

Thm 8.2.3 follows from Thm 8.2.2 and Thm 8.1.3.

We can obtain Thm 8.2.4 by formalizing Thm 8.2.3. (Recall the remark we made after the proof of Thm 8.1.5.) Thm 8.2.5 then follows from this theorem by applying Thm 7.3.2.

Thm 8.2.6 is a logical consequence of Thm 8.2.5 and Thm 7.3.1.

Finally, we get Thm 8.2.7 as a logical consequence of Thm 8.2.2 and Thm 8.2.6.

Thm 8.2.7 is the formalized second incompleteness theorem, and we can obtain the informal second incompleteness theorem by informalizing Thm 8.2.9. We therefore have the following two metatheorems.

**Theorem 8.3.** (Second Incompleteness Theorem)

*If  $S$  is consistent then  $S \not\vdash \text{Consis}(S)$*

**Theorem 8.4.** (Formalized Second Incompleteness Theorem)

$$SA \vdash \text{Consis}(S) \rightarrow \neg \text{Thm}(\text{Consis}(\wedge S), S)$$

#### 8.4. concluding remarks

The purpose of the present paper was to provide a formal axiomatic theory in which one can actually work without resorting to metamathematical arguments. We have set the task of proving Gödel’s (formalized) second incompleteness theorem to test the adequacy of the theory in this respect.

It is obvious that the existing theories like  $\mathbb{P}\mathbb{A}$  (Peano arithmetic) or  $\mathbb{H}\mathbb{A}$  (Heyting arithmetic) are good for studying them but are not good for actually working within them (especially when one has to prove metamathematical theorems like incompleteness theorems). To work within them is as difficult as programming in Turing machines. (We note that Beeson[1] has made a similar remark.)

Since the basic entities one studies in metamathematics are syntactic objects like wffs or proof, and since one must develop some metamathematics within a theory to prove incompleteness theorems, it is desirable that such a theory can handle syntactic objects naturally. It has long been known in computer science that pairing structures provide a natural framework for representing these syntactic objects as a tree structure. These pairing structures are known as McCarthy's symbolic expressions, and are basic objects of the programming language `LISP` (McCarthy[8]). Feferman[4] noticed the usefulness of McCarthy's symbolic expressions and developed two formal theories of symbolic expressions,  $\mathbb{F}\mathbb{M}$  and  $\mathbb{F}\mathbb{M}_0$ , based on second order classical logic. Despite the differences in the logics used and the differences in the basic objects (Feferman uses McCarthy's `sexprs` and we use our `sexprs`), the present work and Feferman's seem to have succeeded in providing workable formal theories in a fairly similar manner. We think that the success owes very much to the mathematical elegance of symbolic expressions.

Another important reason for our choice of `sexprs` as the basic objects in our formal theory  $\mathbb{S}\mathbb{A}$  is that they are implementable on a computer. This is essential because this makes it possible to construct a proof checking system for  $\mathbb{S}\mathbb{A}$  on a computer. Such a proof checking system will not only check if an alleged proof (which is a `sexpr`) is in fact a correct proof, but also will assist in constructing a formal proof. We believe that without such assistance by a computer, it will be impossible to actually construct a formal proof of a reasonably interesting theorem. The proof checking system will be implemented on `Qute`. As we have remarked in Section 2.1, `Qute` is a `PROLOG`-like language which computes functions and relations on `sexprs`, and we can use `Qute` as a theorem prover for formal systems. Moreover, we can define the semantics of `Quto` formally within  $\mathbb{S}\mathbb{A}$ , so that it will be possible to prove properties of programs written in `Qute`. These topics, however, we leave for future publications.

### Acknowledgements

The author is indebted to Dr. Susumu Hayashi for conversations on the material presented in this paper and to Mr. Takafumi Sakurai for implementing Qute.

### References

- [1] Beeson, M. J., Proving programs and programming proofs, *Abstracts of the 7th International Congress of Logic, Methodology and Philosophy of Science*, vol 1, (1983). 3–6.
- [2] Bourne, S. R., *The Unix System*, Addison–Wesley, 1982.
- [3] de Bruijn, N. G., Lambda calculus notation with nameless dummies, A tool for automatic formula manipulation, with application to the Church–Rosser theorem, *Indag. Math.*, 34 (1972) 381–392.
- [4] Feferman, S., Inductively presented system and formalization of meta-mathematics, *Logic Colloquium '80*, North-Holland, 1982.
- [5] Gödel, K., Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, *Monatshefte f. Math. u. Physik*, 38 (1931) 173–198.
- [6] Gordon, M., Milner, R. and Wadsworth, C., *Edinburgh LCF*, Lect. Notes in Comp. Sci. 78, Springer 1978.
- [7] Hayashi, S., Extracting Lisp programs from constructive proofs: A formal theory of constructive mathematics based on Lisp, *Publ. RIMS, Kyoto Univ.* 19 (1983) 169–191.
- [8] McCarthy, J., Recursive functions of symbolic expressions and their computation by machine, Part I, *Comm. ACM*, 3 (1960) 184–195.
- [9] Quine, W. V., *Mathematical Logic (revised version)*, Harvard University Press, 1979.
- [10] Sato, M., Theory of symbolic expressions, I, *Theoretical Computer Science*, 22 (1983) 19–55.
- [11] Sato, M. and Hagiya, M., Hyperlisp, in: J. W. de Bakker and J. C. van Vilet, Eds., *Algorithmic Languages*, North-Holland, (1981). 251–269.
- [12] Sato, M. and Sakurai, T., Qute: A Prolog/Lisp type language for logic programming, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 507–513. 1983.
- [13] Smullyan, R., *Theory of Formal System*, Annals of Mathematics Studies, 47, Princeton University Press, Princeton, 1961.