数理解析研究所講究録
第516巻 1984年 22-39

A Formal Theory of Symbolic Expressions

Masahiko Sato (佐藤雅彦)

Department of Information Science
Faculty of Science
University of Tokyo

Introduction

This paper is an excerpt from the full paper Sato[5], which is in preparation. In this paper we continue our study of the domain **S** of symbolic expressions. In contrast to our former paper Sato[4] (which we will refer to I in the sequel), in this paper, we will study the domain **S** formally within a formal theory **SA** of symbolic expressions.

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 it is 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 $\mathbf{S}_{old}$.

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

Besides our previous works [4, 6], the domain of symbolic expressions recently attracted the attention of some logicians. Feferman[2] introduced second order theories of symbolic expressions and showed that elementary metamathematics can be naturally developed within his systems. Hayashi[3] 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 **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 **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 **BSA** (for *B*asic *S*ymbolic *A*rithmetic). We also explain the intended interpretation of the theory. The theory **BSA** is an adequate theory for developing metamathemaics within it. We refer the reader for our full paper [5] for the details of the development.

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

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 $cons(s, t)$ is a sexp.
3. If s and t are sexps then $snoc(s, t)$ 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 **S**. We denote the image of the function *cons* by **M** and that of *snoc* by **A**. We then have two bijective functions:

$$cons: \mathbf{S}\times\mathbf{S} \to \mathbf{M}$$
$$snoc: \mathbf{S}\times\mathbf{S} \to \mathbf{A}$$

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

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

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

$$car(cons(s, t)) \equiv s$$
$$cdr(cons(s, t)) \equiv t$$

Similarly we define two total functions, *cbr* and *ccr*, on **A** by the equations:

$$cbr(snoc(s, t)) \equiv s$$
$$ccr(snoc(s, t)) \equiv t$$

Compositions of the functions *car*, *cbr*, *ccr* and *cdr* will be abbreviated following the convention in LISP. For instance:

$$cabcdr(t) \equiv car(cbr(ccr(cdr(t))))$$

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

$$[. \ t] \equiv t$$
$$[t_1, \cdots, t_n \ . \ t_{n+1}] \equiv cons(t_1, [t_2, \cdots, t_n \ . \ t_{n+1}]) \ (n \geqq 1)$$
$$[t_1, \cdots, t_n] \equiv [t_1, \cdots, t_n \ . \ *] \ (n \geqq 0)$$

In particular we have

$$[s \ . \ t] \equiv cons(s, t)$$
$$[] \equiv *$$

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

$s[. \ t]$ for $[s \ . \ t]$
$s[t_1, \cdots, t_n \ . \ t_{n+1}]$ for $[s, t_1, \cdots, t_n \ . \ t_{n+1}]$
$s[t_1, \cdots, t_n]$ for $[s, t_1, \cdots, t_n]$

For *snoc*, we only use the following notation

$$(s\ .\ t) \equiv snoc(s,\ t)$$

Parentheses will also be used for grouping. Thus (t) will *not* denote $snoc(t, *)$ but will denote t. (Readers of our previous papers, please forgive our change of notations.)

The basic *induction schema* on **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 $\Phi([s\ .\ t])$

(iii) If $\Phi(s)$ and $\Phi(t)$ then $\Phi((s\ .\ t))$

1.2. symbols and variables

An atom of the form

$$(*\ .\ x)$$

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

$$\rho(\mathrm{a}) \equiv [[*], [*], *, *, *, *, [*]]$$
$$\rho(1) \equiv [*, [*], [*], *, *, *, [*]]$$

We extend ρ homomorphically to Σ^*, i.e., we define $\rho^* : \Sigma^* \to \mathbf{M}$ by $\rho^*(\sigma_1 \cdots \sigma_k) \equiv [\rho(\sigma_1), \cdots, \rho(\sigma_k)]$ $(\sigma_i \in \Sigma)$. 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 π be a name. Then, by definition, π denotes the symbol

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

An atom of the form

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

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 π be such a string. Then, by definition, π denotes the variable

$$(\mathrm{var}\ .\ \rho^*(\pi))$$

We will regard the under score character '_' as a lower case character for convenience.

Example 1.1.

$$\mathrm{var} \equiv (* .\ [*\ .\ [[[*],[*],[*],*,[*],[*],*],\ [[*],[*],*,*,*,*,[*]],\ [[*],[*],[*],*,*,[*],*]]])$$
$$\mathrm{Var} \equiv (\mathrm{var}\ .\ [[[*],*,[*],*,[*],[*],*],\ [[*],[*],*,*,*,*,[*]],\ [[*],[*],[*],*,*,[*],*]])\ \square$$

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[7]. 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 [6].) 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 anyone 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[6].) □

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:

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

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) $\Rightarrow$ *member*$(x, [x \,.\, L])$

(M2) *member*(x, L) $\Rightarrow$ *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

$$(\Gamma)\quad \gamma_1, \cdots, \gamma_n \Longrightarrow \gamma$$

where $n \geqq 0$ and Γ 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 α. (If our set of inductive clauses consist only of (M1) and (M2) above, then α 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 α an informal proof Π of α. According to the extremal clause, α is obtained by applying our inductive clauses finitely many times. Let (Γ) be the last applied clause. Since the clause (Γ) is schematic, when we apply (Γ) we must also specify for each schematic variables x_i a sexp v_i as its value. Let $x_1, \cdots, x_k$ be an enumeration of schematic variables occurring in (Γ) and let

$$\Delta \equiv < x_1 := v_1, \cdots, x_k := v_k >$$

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

$$(\Gamma_\Delta)\quad \alpha_1, \cdots, \alpha_n \Longrightarrow \alpha$$

Note that the conclusion of (Γ_Δ) must be α by our assumption that α is obtained by applying (an instance of) (Γ). That (Γ_Δ) is applicable also means that each α_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 α_i is smaller than that was required to show α, we may assume, as induction hypothesis, that we have an informal proof Π_i of α_i for each $1 \leqq i \leqq n$. Using these data, we can construct a proof Π of α as the figure of the form:

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

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.

$$\frac{\dfrac{}{(\mathrm{M1}) < x := \mathrm{orange},\ L := [\,] >}}{(\mathrm{M2}) < x := \mathrm{orange},\ y := \mathrm{apple},\ L := [\mathrm{orange}] >}$$

□

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 *n*ot *e*qual) which has the property that *ne*(x, y) holds iff x and y are two distinct sexps.

(N1) $\Longrightarrow$ *ne*$(*, [u \,.\, v])$

(N2) $\Longrightarrow$ *ne*$(*, (u \,.\, v))$

(N3) $\Longrightarrow$ *ne*$([s \,.\, t], *)$

(N4) $\Longrightarrow$ *ne*$((s \,.\, t), *)$

(N5) $\Longrightarrow$ *ne*$([s \,.\, t], (u \,.\, v))$

(N6) $\Longrightarrow$ *ne*$((s \,.\, t), [u \,.\, v])$

(N7) *ne*(s, u) $\Longrightarrow$ *ne*$([s \,.\, t], [u \,.\, v])$

(N8) *ne*(t, v) $\Longrightarrow$ *ne*$([s \,.\, t], [u \,.\, v])$

(N9) *ne*(s, u) $\Longrightarrow$ *ne*$((s \,.\, t), (u \,.\, v))$

(N10) $ne(t, v) \Rightarrow ne((s \,.\, t), (u \,.\, v))$

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

(A1) $\Rightarrow assoc(x, [[x \,.\, v] \,.\, L], v)$

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

Example 2.2.

assoc(c, [[a . apple], [b . banana], [c . carrot]], carrot *)* □

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

(G1) $\Rightarrow get(*, [v \,.\, L], v)$

(G2) $get(i, L, v) \Rightarrow get([* \,.\, i], [w \,.\, L], v)$

Example 2.3.

get([*, *], [lisp, prolog, qute], qute *)* □

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) $assoc((\text{var} \,.\, t), Env, v) \Rightarrow eval((\text{var} \,.\, t), Env, v)$

(E2) $\Rightarrow eval(*, Env, *)$

(E3) $eval(s, Env, u), eval(t, Env, v) \Rightarrow eval([s \,.\, t], Env, [u \,.\, v])$

(E4) $eval(s, Env, u), eval(t, Env, v) \Rightarrow eval((\text{snoc} \,.\, [s, t]), Env, (u \,.\, v))$

(E5) $\Rightarrow eval((* \,.\, t), Env, (* \,.\, t))$

(E6) $\Rightarrow eval((\text{quote} \,.\, t), Env, t)$

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)] *)* □

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

(L1) $\Rightarrow lproves([], [], FS)$

(L2) $proves(p, a, FS), lproves(P, A, FS) \Rightarrow lproves([p \,.\, P], [a \,.\, A], FS)$

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

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

(T1) $proves(p, a, FS) \Rightarrow theorem(a, FS)$

We show by an example how our intuitive idea of proof has been formalized. Recall that the relation *assoc* was defined by the two clause (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 system [*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:

```
[ assoc
        , [[x, [[x . v] . L], v]]
        , [[x, [[y . w] . L], v]
                , ne[x, y]
                , assoc[x, L, v]]
        ]
```

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:

$$\gamma_1, \cdots, \gamma_n \implies \gamma$$

and that the general form of γ or γ_i is:

$$\mathit{Prd}(\mathit{Arg}_1, \cdots, \mathit{Arg}_k)$$

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 translated into $*$. If *Arg* is of the form (iii), its translation is $[\alpha^* \ . \ \beta^*]$ where α^* (β^*) is the translation of α (β, resp.). Similarly, but slightly differently, case (iv) is translated into $(: \alpha^* \ . \ \beta^*)$. (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*.

```
+ assoc
        | x, [[x . v] . L], v
        | x, [[y . w] . L], v
                - ne[x, y]
                - assoc[x, L, v]
        ;
```

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 :

```
[ [member, [*], [[x . orange], [y . apple], [L . [orange]]]],
        [ [member, *, [[x . orange], [L . []]]] ] ]
```

Let *Member* be the following sexp:

```
+ member
        | x, [x . L]
        | x, [y . L]
                - member[x, L]
        ;
```

Then we can easily verify that

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

holds and hence

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

holds. □

2.4. universal formal system

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

Univ ≡ [*Ne*, *Assoc*, *Get*, *Eval*, *Lproves*, *Proves*, *Theorem*]

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

```
+ ne
      |*, [u . v]
      |*, (: u . v )
      |[s . t], *
      |(: s . t ), *
      |[s . t], (: u . v )
      |(: s . t ), [u . v]
      |[s . t], [u . v]
            - ne[s, u]
      |[s . t], [u . v]
            - ne[t, v]
      |(: s . t ), (: u . v )
            - 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]
                - get[i, R, [c . C]]
                - eval[c, Env, a]
                - eval[C, Env, A]
                - lproves[P, A, FS]
        ;

+ theorem
        | a, FS
                - proves[p, a, FS]
        ;
```

The following theorem establishes that **Univ** is in fact a universal formal system.

Theorem 2.1.

(i) $ne(x, y) \iff \vdash_{\mathbf{Univ}} \mathrm{ne}[x, y]$

(ii) $assoc(x, L, v) \iff \vdash_{\mathbf{Univ}} \mathrm{assoc}[x, L, v]$

(iii) $get(i, L, v) \iff \vdash_{\mathbf{Univ}} \mathrm{get}[i, L, v]$

(iv) $eval(t, E, v) \iff \vdash_{\mathbf{Univ}} \mathrm{eval}[t, E, v]$

(v) $lproves(P, A, FS) \iff \vdash_{\mathbf{Univ}} \mathrm{lproves}[P, A, FS]$

(vi) $proves(p, a, FS) \iff \vdash_{\mathbf{Univ}} \mathrm{proves}[p, a, FS]$

(vii) $theorem(a, FS) \iff \vdash_{\mathbf{Univ}} \mathrm{theorem}[a, FS]$

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

Corollary 2.2.

(i) $p \vdash_{FS} a \iff \vdash_{\mathbf{Univ}} \mathrm{proves}[p, a, FS]$

(ii) $\vdash_{FS} a \iff \vdash_{\mathbf{Univ}} \mathrm{theorem}[a, FS]$

3. Formal theory of symbolic expressions: BSA

In this section we introduce a formal theory of symbolic expressions which we call **BSA** (for *B*asic *S*ymbolic *A*rithmetic). 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 ***BSA*** to denote the theory which we now define by a traditional method.

3.1. language

The language of *BSA* consists of the following symbols.

- *individual symbols* : **nil**
- *function symbols* : **cons**, **snoc**
- *pure variables* : $\mathbf{var}_t$ 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 *BSA*. We first define *variables* as follows.

1. For each sexp t, the pure variable $\mathbf{var}_t$ is a variable.
2. If x is a variable then $\mathbf{free}(x)$ 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 $\mathbf{free}(x)$ 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 $\mathbf{cons}(s, t)$ and $\mathbf{snoc}(s, t)$ are terms.

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

1-2. If s and t are terms then $\mathbf{eq}(s, t)$ and $\mathbf{lt}(s, t)$ are wffs.

3-4. If $a_1, \cdots, a_n$ $(n \geqq 0)$ are wffs then $\mathbf{and}(a_1, \cdots, a_n)$ and $\mathbf{or}(a_1, \cdots, a_n)$ are wffs.

5. If $a_1, \cdots, a_n$ $(n \geqq 0)$ and b are wffs then $\mathbf{imply}((a_1, \cdots, a_n), b)$ is a wff.

6-7. If $x_1, \cdots, x_n$ $(n \geqq 0)$ are distinct pure variables and a is a wff then $\mathbf{all}((x_1, \cdots, x_n), a)$ and $\mathbf{exist}((x_1, \cdots, x_n), a)$ 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 $\mathbf{free}(x)$
$s = t$ for $\mathbf{eq}(s, t)$
$s < t$ for $\mathbf{lt}(s, t)$
$s \leqq t$ for $\mathbf{or}(\mathbf{lt}(s, t), \mathbf{eq}(s, t))$

$a_1 \wedge \cdots \wedge a_n$ for **and**$(a_1, \cdots, a_n)$

$a_1 \vee \cdots \vee a_n$ for **or**$(a_1, \cdots, a_n)$

$a_1, \cdots, a_n \rightarrow b$ for **imply**$((a_1, \cdots, a_n), b)$

$a \leftrightarrow b$ for **and**(**imply**$((a), b)$, **imply**$((b), a)$)

$\forall(x_1, \cdots, x_n; a)$ for **all**$((x_1, \cdots, x_n), a)$

$\exists(x_1, \cdots, x_n; a)$ for **exist**$((x_1, \cdots, x_n), a)$

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 result 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**$(a_1, \cdots, a_n)$ (**or**$(a_1, \cdots, a_n)$) and e_i $(1 \leqq i \leqq n)$ is the result of substituting t for x in a_i then e is **and**$(e_1, \cdots, e_n)$ (**or**$(e_1, \cdots, e_n)$, resp.).

II.5. If d is **imply**$((a_1, \cdots, s_n), b)$, e_i $(1 \leqq i \leqq n)$ 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**$((e_1, \cdots, e_n), c)$.

II.6. If d is **all**$((x_1, \cdots, x_n), a)$, u (y) is the elevation of t (x, resp.) with respect to the sequence of pure variables $x_1, \cdots, x_n$ and b is the result of substituting u for y in a then e is **all**$((x_1, \cdots, x_n), b)$

II.7. If d is **exist**$((x_1, \cdots, x_n), a)$, u (y) is the elevation of t (x, resp.) with respect to the sequence of pure variables $x_1, \cdots, x_n$ and b is the result of substituting u for y in a then e is **exist**$((x_1, \cdots, x_n), b)$

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

1.1. If t is a variable whose pure part is x_i for some i $(1 \leqq i \leqq n)$ then u is **free**(t).

1.2. If t is a variable whose pure part does not appear in the sequence $x_1, \cdots, x_n$ 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 $x_1, \cdots, x_n$ 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 $x_1, \cdots, x_n$ 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 $\exists(x,\ y;\ z=\mathbf{cons}(x,\ y))$. Then the result of substituting the term $\mathbf{cons}(x,\ y)$ for z in this wff is calculated similarly as above and we obtain the wff $\exists(x,\ y;\ \mathbf{cons}(\#x,\ \#y)=\mathbf{cons}(x,\ y))$. □

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 [1]. □

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

We say that a variable x *occurs free in* a designator d if $d_x[\mathbf{nil}]$ 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 gives 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 $t_1, \cdots, t_n$ be a sequence of terms, $x_1, \cdots, x_n$ be a sequence of variables whose pure parts are distinct and d be a designator. We can define the result of simultaneously bind substituting $t_1, \cdots, t_n$ for $x_1, \cdots, x_n$ in d similarly as above, and we use the notation $d_{x_1,\cdots,x_n}[\![t_1, \cdots, t_n]\!]$ for it.

3.5. proofs

We formulate our formal 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:

$$\frac{a_1 \quad \cdots \quad a_n}{a} \qquad n\geqq 0$$

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 $a_1, \cdots, a_n$ 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.

$$(\wedge I)\quad \frac{a_1 \quad \cdots \quad a_n}{a_1 \wedge \cdots \wedge a_n} \qquad (\wedge E)_i\quad \frac{a_1 \wedge \cdots \wedge a_n}{a_i} \quad 1 \leqq i \leqq n$$

$$(\vee I)_i\quad \frac{a_i}{a_1 \vee \cdots \vee a_n} \quad 1 \leqq i \leqq n \qquad (\vee E)\quad \frac{a_1 \vee \cdots \vee a_n \quad \begin{matrix}[\,a_1\,] \\ \cdots \\ c\end{matrix} \quad \cdots \quad \begin{matrix}[\,a_n\,] \\ \cdots \\ c\end{matrix}}{c}$$

$$(\rightarrow I)\quad \frac{\begin{matrix}[\,a_1\,] \quad \cdots \quad [\,a_n\,] \\ b\end{matrix}}{a_1, \cdots, a_n \rightarrow b} \qquad (\rightarrow E)\quad \frac{a_1, \cdots, a_n \rightarrow b \quad a_1 \quad \cdots \quad a_n}{b}$$

$$(\forall I)\quad \frac{a_{x_1, \cdots, x_n}[\![y_1, \cdots, y_n]\!]}{\forall(x_1, \cdots, x_n;\ a)} \qquad (\forall E)\quad \frac{\forall(x_1, \cdots, x_n;\ a)}{a_{x_1, \cdots, x_n}[\![t_1, \cdots, t_n]\!]}$$

$$(\exists I)\quad \frac{a_{x_1, \cdots, x_n}[\![t_1, \cdots, t_n]\!]}{\exists(x_1, \cdots, x_n;\ a)} \qquad (\exists E)\quad \frac{\exists(x_1, \cdots, x_n;\ a) \quad \begin{matrix}[\,a_{x_1, \cdots, x_n}[\![y_1, \cdots, y_n]\!]\,] \\ b\end{matrix}}{b}$$

$$(=)\quad \frac{}{t = t} \qquad (=subst)\quad \frac{a_{x_1, \cdots, x_n}[\,s_1, \cdots, s_n\,] \quad s_1 = t_1 \quad \cdots \quad s_n = t_n}{a_{x_1, \cdots, x_n}[\,t_1, \cdots, t_n\,]}$$

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

Note that we may identify the wffs **and**() and **or**() with 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 **or**(), $\neg a$ 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.

$$(cons \neq nil)\quad \frac{\mathbf{cons}(s,\ t) = \mathbf{nil}}{\perp} \qquad (snoc \neq nil)\quad \frac{\mathbf{snoc}(s,\ t) = \mathbf{nil}}{\perp}$$

$$(cons \neq snoc)\quad \frac{\mathbf{cons}(s,\ t) = \mathbf{snoc}(u,\ v)}{\perp}$$

$$(cons = cons)_i\quad \frac{\mathbf{cons}(s_1,\ s_2) = \mathbf{cons}(t_1,\ t_2)}{s_i = t_i} \quad i = 1, 2$$

$$(snoc{=}snoc)_i \quad \frac{\mathbf{snoc}(s_1,\ s_2) = \mathbf{snoc}(t_1,\ t_2)}{s_i = t_i} \qquad i{=}1,2$$

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

$$(<*) \quad \frac{r < *}{\bot} \qquad (<snoc) \quad \frac{r < \mathbf{snoc}(s,\ t)}{\bot}$$

$$(<)_i \quad \frac{}{t_i < \mathbf{cons}(t_1,\ t_2)} \qquad i{=}1,2 \qquad (<cons)_i \quad \frac{s < t_i}{s < \mathbf{cons}(t_1,\ t_2)} \qquad i{=}1,2$$

$$(<consE) \quad \frac{r < \mathbf{cons}(s,\ t) \qquad \begin{matrix}[\ r = s\ ] \\ c\end{matrix} \qquad \begin{matrix}[\ r < s\ ] \\ c\end{matrix} \qquad \begin{matrix}[\ r = t\ ] \\ c\end{matrix} \qquad \begin{matrix}[\ r < t\ ] \\ c\end{matrix}}{c}$$

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

$$(ind) \quad \frac{a_z[\mathbf{nil}] \qquad \begin{matrix}[\ a_z[x]\ ]\ [\ a_z[y]\ ] \\ a_z[\mathbf{cons}(x,\ y)]\end{matrix} \qquad \begin{matrix}[\ a_z[x]\ ]\ [\ a_z[y]\ ] \\ a_z[\mathbf{snoc}(x,\ y)]\end{matrix}}{a_z[t]}$$

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 any assumption other than the induction hypotheses on which the premises $a_z[\mathbf{cons}(x,\ y)]$ and $a_z[\mathbf{snoc}(x,\ y)]$ 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* $[\![t]\!]$ of a closed term t as follows.

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

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 $a_1, \cdots, a_n$ $(n \geqq 0)$ are primitive wffs then $a_1 \wedge \cdots \wedge a_n$ and $a_1 \vee \cdots \vee a_n$ are primitive wffs.

5. If $a_1, \cdots, a_n$ $(n \geqq 0)$ and b are primitive wffs then $a_1, \cdots, a_n \rightarrow b$ is a primitive wff.

6-7. If $x_1, \cdots, x_n$ is a sequence of distinct pure variables, $t_1, \cdots, t_n$ is a sequence of terms, u_i $(1 \leqq i \leqq n)$ is the elevation of t_i with respect to $x_1, \cdots, x_n$ and a is a primitive wff then $\forall(x_1, \cdots, x_n; x_1 < u_1, \cdots, x_n < u_n \rightarrow a)$ and $\exists(x_1, \cdots, x_n; x_1 < u_1 \wedge \cdots \wedge x_n < u_n \wedge a)$ are primitive wffs.

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

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

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 Σ-*wffs* as follows:

1. A primitive wff is a Σ-wff.

2-3. If $a_1, \cdots, a_n$ $(n \geqq 0)$ are Σ-wffs then $a_1 \wedge \cdots \wedge a_n$ and $a_1 \vee \cdots \vee a_n$ are Σ-wffs.

4. If $a_1, \cdots, a_n$ $(n \geqq 0)$ are primitive wffs and b is a Σ-wff then $a_1, \cdots, a_n \rightarrow b$ is a Σ-wff.

5. If $x_1, \cdots, x_n$ is a sequence of distinct pure variables and a is a Σ-wff then $\exists(x_1, \cdots, x_n; a)$ is a Σ-wff.

We can define the *truth* of a closed Σ-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 Σ-wff $\exists(x_1, \cdots, x_n; a)$ is defined to be *true* if we can find a sequence of closed terms $t_1, \cdots, t_n$ for which $a_{x_1, \cdots, x_n}[t_1, \cdots, t_n]$ becomes *true*.

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

3.7. BSA as a formal system

We now define **BSA** as a formal system and then define an isomorphism from *BSA* to **BSA**. It is possible to regard this isomorphism as an (symbolic) arithmetization of *BSA*. Here we will not define the concept of proof in **BSA** since we give a full description of **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:

$\mathbf{BSA}_0 \equiv$ [*Ne, Non_member, Pure_variable, Pure_variable_list, Variable, Term, Wff, Wff_list*]

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

Example 3.2.

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

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

We now explain that the concepts which we defined formally here are essentially the same as the corresponding concepts which we defined for *BSA*. To this end we define a translation from syntactic objects like terms or wffs in *BSA* into **S**. We denote the translation of $\boldsymbol{d}$ by $\boldsymbol{d}^\dagger$.

Terms in *BSA* are translated as follows.

1.1. $\mathbf{var}_t{}^\dagger$ is (var . t).

1.2. $\mathbf{free}(\boldsymbol{x})^\dagger$ is (free . $\boldsymbol{x}^\dagger$).

2. $\mathbf{nil}^\dagger$ is *.

3-4. $\mathbf{cons}(s, t)^\dagger$ is [$s^\dagger$. $\boldsymbol{t}^\dagger$] and $\mathbf{snoc}(s, \boldsymbol{t})^\dagger$ is (: $s^\dagger$. $\boldsymbol{t}^\dagger$).

The translation of wffs in *BSA* is defined as follows.

1-2. $\mathbf{eq}(s, t)^\dagger$ is eq[$s^\dagger$, $\boldsymbol{t}^\dagger$] and $\mathbf{lt}(s, t)^\dagger$ is lt[$s^\dagger$, $\boldsymbol{t}^\dagger$].

3-4. $\mathbf{and}(\boldsymbol{a}_1, \cdots, \boldsymbol{a}_n)^\dagger$ is and[$\boldsymbol{a}_1{}^\dagger, \cdots, \boldsymbol{a}_n{}^\dagger$] and $\mathbf{or}(\boldsymbol{a}_1, \cdots, \boldsymbol{a}_n)^\dagger$ is or[$\boldsymbol{a}_1{}^\dagger, \cdots, \boldsymbol{a}_n{}^\dagger$].

5. $\mathbf{imply}((\boldsymbol{a}_1, \cdots, \boldsymbol{a}_n), \boldsymbol{b})^\dagger$ is imply[[$\boldsymbol{a}_1{}^\dagger, \cdots, \boldsymbol{a}_n{}^\dagger$], $\boldsymbol{b}^\dagger$].

6-7. $\mathbf{all}((\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n), \boldsymbol{a})^\dagger$ is all[(abs . [[$\boldsymbol{x}_1{}^\dagger, \cdots, \boldsymbol{x}_n{}^\dagger$], $\boldsymbol{a}^\dagger$])] and $\mathbf{exist}((\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n), \boldsymbol{a})^\dagger$ is ex[(abs . [[$\boldsymbol{x}_1{}^\dagger, \cdots, \boldsymbol{x}_n{}^\dagger$], $\boldsymbol{a}^\dagger$])].

It is then easy to verify that this translation sends each syntactic entity in *BSA* into corresponding entity in **BSA**. Thus if $\boldsymbol{a}$ is a wff in the sense of *BSA* then $\boldsymbol{a}^\dagger$ is a wff in **BSA**, that is, we have $\vdash_{\mathbf{BSA}_0}$wff[$\boldsymbol{a}^\dagger$]. Moreover for each wff a in **BSA** we can uniquely find a wff $\boldsymbol{a}$ in *BSA* such that $\boldsymbol{a}^\dagger$ 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 *BSA* and **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 *BSA* as abbreviations for the corresponding objects in **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 sexp as a wff, we will use syntactic variables $\boldsymbol{a}$, $\boldsymbol{b}$ or $\boldsymbol{c}$ for it.

Example 3.3.

∀(x; ∃(x; x = #x)) is an abbreviation of the sexp

all[(abs . [[x], ex[(abs . [[x], eq[x, (free . x)]])]])]

which is a wff in **BSA**. □

References

[1] 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.

[2] Feferman, S.: Inductively presented system and formalization of meta-mathematics, *Logic Colloquium '80*, North-Holland, 1982.

[3] 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.

[4] Sato, M.: Theory of symbolic expressions, I, *Theoretical Computer Science* **22** (1983) 19-55.

[5] Sato, M.: Theory of symbolic expressions, II, *in preparation.*

[6] 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.

[7] Smullyan, R.: *Theory of Formal System*, Annals of Mathematics Studies, **47**, Princeton University Press, Princeton, 1961.