九州大学学術情報リポジトリ
Kyushu University Institutional Repository

Introducing Types into Elementary Formal Systems

Kiwata, Kazuhiro
Research Institute of Fundamental Information Science Kyushu University

Arikawa, Setsuo
Research Institute of Fundamental Information Science Kyushu University

https://hdl.handle.net/2324/3183

出版情報：RIFIS Technical Report. 84, 1994-04-20. Research Institute of Fundamental Information Science, Kyushu University
バージョン：
権利関係：

RIFIS-TR-CS-84

RIFIS Technical Report

Introducing Types into Elementary Formal Systems

Kazuhiro Kiwata
Setsuo Arikawa

April 20, 1994

Research Institute of Fundamental Information Science
Kyushu University 33
Fukuoka 812, Japan
E-mail: arikawa@rifis.kyushu-u.ac.jp Phone: 092-641-1101 ex. 4484

Introducing Types into Elementary Formal Systems

Kazuhiro Kiwata and Setsuo Arikawa
Research Institute of Fundamental Information Science,
Kyushu University 33, Fukuoka 812, Japan
e-mail: {kiwata, arikawa}@rifis.kyushu-u.ac.jp

Abstract

The elementary formal systems (EFS's for short) work as a theoretical basis in the various fields of computer science such as formal language theory, programming semantics, machine learning and so on. In this paper, we first define proof figures which illustrate proofs of EFS and show standard proof figures for ground atoms which are distinctive in the usage of inference rules. Then we formalize typed EFS, which is a natural extension of EFS obtained by replacing variables by typed variables and show some basic properties as a logic programming language. Finally, we illustrate the first order logic in term of our typed EFS's.

1 Introduction

Types, i.e., data types, are sets of data in programming languages and are useful to verify the consistency of programs. Type theory is a tool to investigate types mathematically. Especially type theory is extensively studied in the formal system of λ-calculus which is a model of functional programming languages [4, 9]. We can also see types as logical formulas. Consequently we can relate the theory of programs such as specification, verification, synthesis, and extraction to constructive mathematics [7, 5].

The elementary formal system (EFS for short) is a formal system which was invented by Smullyan [8] to develop his recursive function theory. It is shown that EFS gives a theoretical foundation to various fields of computer science [2]. For example, in formal language theory, there is a correspondence between Chomsky hierarchy and EFS classes. We also regard EFS as a logic program so that we can use the refutation procedure of logic programs as a procedure to accept languages. Recently it is shown that EFS's are a unifying framework for learning, especially for inductive inference of languages [2].

In this paper, we focus on the property of EFS's as a logic programming language and introduce types as sets of data into EFS's. In section 2, we give definitions of concepts on EFS's necessary for our discussion. In section 3, we define proof figures which illustrate the provable relation of EFS's and show there exist standard proof figures for ground atoms. In section 4, we first formalize typed EFS's and show that it has important properties as a logic programming language like the ordinary EFS's. Finally we illustrate the first order logic in the framework of our typed EFS's.

2 EFS

First we prepare some concepts on EFS's. For detailed definitions on first order logic, logic programming and EFS's, readers should refer to [6, 1, 2].

Let Σ be a finite *alphabet*, X be a countable set of *variables* and Π be a finite set of *predicate symbols*. A finite non-empty string of symbols in $(\Sigma \cup X)$ is called a *pattern*, which stands for a term in EFS's. *Atomic formulas*(atoms for short), *clauses*, *ground clauses*, *empty clauses*, *substitutions* and *unifiers* are defined in the same way as in the logic programming.

Definition 1 An *EFS(Elementary Formal System)* S is a triplet (Σ, Π, Γ), where Γ is a finite set of definite clauses. A definite clause is a clause of the form

$$A \leftarrow B_1, \ldots, B_n \ (n \geq 0).$$

The definite clauses in Γ are also called *axioms* of S.

Definition 2 Let $S = (\Sigma, \Pi, \Gamma)$ be an EFS. We define the relation $\Gamma \vdash C$ for a clause C of S inductively as follows:

(1) If $\Gamma \ni C$, then $\Gamma \vdash C$.

(2) If $\Gamma \vdash C$, then $\Gamma \vdash C\theta$ for any substitution θ.

(3) If $\Gamma \vdash A \leftarrow B_1, \ldots, B_{n+1}$ and $\Gamma \vdash B_{n+1}$, then $\Gamma \vdash A \leftarrow B_1, \ldots, B_n$.

C is *provable* from Γ if $\Gamma \vdash C$.

Definition 3 Let $S = (\Sigma, \Pi, \Gamma)$ be an EFS, p be a predicate symbol with arity n in Π. Then we define

$$L(S,p) = \{ \ (\pi_1, \ldots, \pi_n) \in (\Sigma^+)^n \mid \Gamma \vdash p(\pi_1, \ldots, \pi_n) \leftarrow \ \}.$$

In case $n = 1$, $L(S,p)$ is a language over Σ. A language $L \subseteq \Sigma^+$ is *definable by EFS* or an *EFS language* if there exist S and p such that $L = L(S,p)$.

Let E be a term, an atom or a clause. Then $v(E)$ denotes the set of all variables in E.

Definition 4 A definite clause $A \leftarrow B_1, \ldots, B_n$ is *variable-bounded* if $v(A) \supseteq v(B_i)$ $(i = 1, \ldots, n)$, and an EFS is variable-bounded if its axioms are all variable-bounded.

A derivation is defined as follows. We assume a computation rule R to select an atom from a goal.

Definition 5 Let S be an EFS, G be a goal of S, and R be a computation rule. A *derivation* from G is a (finite or infinite) sequence of triplets (G_i, θ_i, C_i) $(i = 0, 1, \ldots)$ which satisfies the following conditions.

(1) G_i is a goal, θ_i is a substitution, C_i is a variant of an axiom of S, and $G_0 \equiv G$.

(2) $v(C_i) \cap v(C_j) = \emptyset$ $(i \neq j)$, and $v(C_i) \cap v(G) = \emptyset$ for every i.

(3) If G_i is $\leftarrow A_1, \ldots, A_k$ and A_m is the atom selected by R, then C_i is $A \leftarrow B_1, \ldots, B_q$, θ_i is a unifier of A and A_m, and G_{i+1} is a goal

$$(\leftarrow A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta_i.$$

G_{i+1} is called a *resolvent* of G_i and C_i by θ_i. A *refutation* is a finite derivation ending with the empty goal $\square$, and if $G_n = \square$, then we call n the *length* of the refutation.

We give the semantics of logic programming languages on the Herbrand base and the semantics of EFS's as follows. The Herbrand base is the set of all ground atoms, denoted by $B(S)$. We introduce two sets which give the semantics of EFS's.

$$SS(S) = \{A \in B(S) \mid \textit{There is a refutation from} \leftarrow A\}$$
$$PS(S) = \{A \in B(S) \mid \Gamma \vdash A \leftarrow\}$$

3 Proof Figures for EFS

In this section, we investigate proofs of EFS's. First, we define proof figures for EFS's, which represent the relation $\vdash$ as figures. Proof figures whose conclusions are a clause C are denoted by Π_C, Π'_C,and so on.

Definition 6 Let $S = (\Sigma, \Pi, \Gamma)$ be an EFS, and C be a clause of S such that $\Gamma \vdash C$. A *proof figure* Π_C of C and the length $length(\Pi_C)$ of Π_C are defined inductively as follows:

(1) If $\Gamma \ni C$, then the proof figure of C and its length are C, 1 respectively.

(2) If there exist a clause C' and a substitution θ such that $\Gamma \vdash C'$ and $C \equiv C'\theta$, then the proof figure of C and its length are

$$\frac{\Pi_{C'}}{C}\ (\mathrm{SUB}),$$

$$length(\Pi_C) = length(\Pi_{C'}) + 1.$$

(3) If there exist a clause $C' \equiv A \leftarrow B_1, \ldots, B_{n+1}$ such that $\Gamma \vdash C'$, $C'' \equiv B_{n+1}$ such that $\Gamma \vdash C''$ and $C \equiv \leftarrow B_1, \ldots, B_n$, then the proof figure of C and its length are

$$\frac{\Pi_{C'} \quad \Pi_{C''}}{C}\ (\mathrm{MP}) \quad or \quad \frac{\Pi_{C''} \quad \Pi_{C'}}{C}\ (\mathrm{MP}),$$

$$length(\Pi_C) = \max\{length(\Pi_{C'}), length(\Pi_{C''})\} + 1.$$

Let $\theta = \{x_1 := \tau_1, \ldots, x_n := \tau_n\}$ be a substitution. Then a *support* $D(\theta)$ of θ is the set $\{x_1, \ldots, x_n\}$.

Definition 7 Let $S = (\Sigma, \Pi, \Gamma)$ be an EFS, and C be a clause of S such that $\Gamma \vdash C$. Then a *normalized proof figure* of C is a proof figure which satisfies the following conditions.

(1) Every substitution used for the inference rule SUB is a ground substitution whose support is equal to the set of variables of the clause to which the substitution is applied

(2) The inference rule MP is only applied to gound clauses.

Theorem 1 Let $S = (\Sigma, \Pi, \Gamma)$ be a variable-bounded EFS, and A be a ground atom of S. If $\Gamma \vdash A \leftarrow$, then there exists a normalized proof figure of $A \leftarrow$.

Proof. We prove this theorem by an induction on the length of proof figures. If $n = 1$, then $\Gamma \ni A \leftarrow$, that is, A itself is a proof figure. Clearly this is a normalized proof figure. If $n = 2$, then there are the following two cases.

a) There exist an axiom $A' \leftarrow$ of S and a substitution θ such that $A \equiv A'\theta$ and the proof figure is Fig. 1.

b) There exist clauses $A \leftarrow B_1$ and $B_1 \leftarrow$ such that they are ground instances of axioms of S and the proof figure is Fig. 2.

$$\frac{A' \leftarrow}{A \leftarrow}\ (\mathrm{SUB}) \qquad \text{Fig. 1}$$

$$\frac{A \leftarrow B_1 \quad B_1 \leftarrow}{A \leftarrow}\ (\mathrm{MP}) \qquad \text{Fig. 2}$$

Clearly these are normalized proof figures.

Suppose the hypothesis of induction holds for $n \leq k$ $(k \geq 2)$, and then consider $n = k+1$. If MP is not used in the proof figure, the theorem clearly holds. Now suppose MP is used in the proof figure at least once. Then there are two cases about the last columns of the proof figures, where $AandB_1$ are ground.

$$\dfrac{\dfrac{\vdots \quad \vdots}{A' \leftarrow B'_1 \quad B'_1 \leftarrow}}{\dfrac{A' \leftarrow}{A \leftarrow}}$$

Fig. 3

$$\dfrac{\vdots \quad \vdots}{A \leftarrow B_1 \quad B_1 \leftarrow}$$
$$A \leftarrow$$

Fig. 4

If $A \equiv A'\theta$, then Fig. 3 is transformed to Fig. 5. $A'\theta, B'_1\theta$ are ground so that Fig. 5 can be reduced to Fig. 4. Fig. 5 is not longer on the length than Fig. 3.

$$\dfrac{\dfrac{\vdots}{A' \leftarrow B'_1} \quad \dfrac{\vdots}{B'_1 \leftarrow}}{\dfrac{A'\theta \leftarrow B'_1\theta \quad B'_1\theta \leftarrow}{A \leftarrow}}$$

Fig. 5

If the same procedure is applied to $A \leftarrow B_1$, then finally we can get Fig. 6, where $A, B_1, \ldots, B_m$ are ground and $A' \leftarrow B'_1, \ldots, B'_m$ is an axiom of S $(m \geq 1)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{A' \leftarrow B'_1, \ldots, B'_m}{A \leftarrow B_1, \ldots, B_m} \quad \dfrac{\vdots}{B_m \leftarrow}}{A \leftarrow B_1, \ldots, B_{m-1}} \quad \dfrac{\vdots}{B_{m-1} \leftarrow}}{A \leftarrow B_1, \ldots, B_{m-2}}}{\dfrac{\dfrac{\vdots}{A \leftarrow B_1} \quad \dfrac{\vdots}{B_1 \leftarrow}}{A \leftarrow}}$$

Fig. 6

Let $\Pi_{A\leftarrow}$ be the proof figure, Fig. 6, of A, $\Pi_{B_1\leftarrow}, \ldots, \Pi_{B_m\leftarrow}$ be the proof figures of $B_1, \ldots, B_m$ in $\Pi_{A\leftarrow}$. Then $length(\Pi_{B_i\leftarrow}) < length(\Pi_{A\leftarrow}) \leq k+1$ $(1 \leq i \leq m)$. Hence from the hypothesis of induction, there exist a normalized proof figure of $B_i \leftarrow$. Therefore we can get a normalized proof figure of $A \leftarrow$ from $\Pi_{A\leftarrow}$. □

From the proof of Theorem 1, the normalized proof figure is given as the shortest proof figure of a ground atom A. For ground cluases, we can also prove the existence of the normalized proof figure.

Example 1 Let $S = (\{a, b\}, \{p\}, \Gamma)$ be an EFS with

$$\Gamma = \left\{ \begin{array}{l} p(a) \leftarrow , \\ p(bxy) \leftarrow p(x), p(y) \end{array} \right\}.$$

According to the procedure in the proof of Theorem 1, we can construct a normalized proof figure of $p(baa) \leftarrow$ (Fig. 8) from its proof figure (Fig. 7). The lengths of those figures are 4

and 5, respectively.

$$\dfrac{\dfrac{\dfrac{p(bxy) \leftarrow p(x), p(y)}{p(bxa) \leftarrow p(x), p(a)}\,(\mathrm{SUB}) \qquad p(a) \leftarrow}{p(bxa) \leftarrow p(x)}\,(\mathrm{MP}) \qquad}{\dfrac{p(baa) \leftarrow p(a)}{p(baa) \leftarrow}\,(\mathrm{SUB}) \qquad p(a) \leftarrow}\,(\mathrm{MP})$$

Fig. 7

$$\dfrac{\dfrac{\dfrac{p(bxy) \leftarrow p(x), p(y)}{p(baa) \leftarrow p(a), p(a)}\,(\mathrm{SUB}) \qquad p(a) \leftarrow}{p(baa) \leftarrow p(a)}\,(\mathrm{MP}) \qquad}{p(baa) \leftarrow}\,(\mathrm{MP})$$

Fig. 8

4 Typed EFS

In this section, we formalize the new concept called a typed EFS. First we define types and typed variables.

Definition 8 *Types*, denoted by $\mathcal{T}, \mathcal{T}_1, \ldots$, are subsets of Σ^+ which satisfies the following condition $(*)$.

$$\text{For all } \tau \in \Sigma^+, \text{ it is decidable whether } \tau \in \mathcal{T} \text{ or not. } (*)$$

Let x be a variable and $\mathcal{T}$ be a type. Then $x : \mathcal{T}$ is called a *typed variable*, which means type of x is $\mathcal{T}$. A set $\{x_1 : \mathcal{T}_1, \ldots, x_n : \mathcal{T}_n\}$ is called a *context*, denoted by C. C is *consistent* iff it satisfies the following condition.

$$\text{If } x_i \equiv x_j, \text{ then } \mathcal{T}_i = \mathcal{T}_j.$$

Hence a variable x in an ordinary EFS is represented by $x : \Sigma^+$. Now we define typed terms and typed definite clauses.

Definition 9 A *typed term* E is a pair $(E', \{x_1 : \mathcal{T}_1, \ldots, x_n : \mathcal{T}_n\})$, where E' is a term, $\{x_1, \ldots, x_n\} = v(E')$, and $\{x_1 : \mathcal{T}_1, \ldots, x_n : \mathcal{T}_n\}$, denoted by $C(E')$, is consistent. A *typed definite clause* is defined in the same way. We also call E' itself a typed term (typed definite clause).

For $\mathcal{T}_1, \ldots, \mathcal{T}_n$ in $C(E')$, $\mathcal{T}_1 = \Sigma^+, \ldots, \mathcal{T}_n = \Sigma^+$, then E is an ordinary definite term(cluase).

We formalize the substitution for typed terms(typed definite clauses) as follows.

Definition 10 Let $\tau_1 = (\tau_1', C(\tau_1')), \ldots, \tau_k = (\tau_k', C(\tau_k'))$ be typed terms, where $C(\tau_1') \cup \cdots \cup C(\tau_k')(= C)$ is consistent. Then we call $(\{x_1 := \tau_1', \ldots, x_k := \tau_k'\}, C)$ a *substitution.*

If $C = \{z_1 : \Sigma^+, \ldots, z_n : \Sigma^+\}$, then the substitution is a substitution in the sense of logic programming.

Example 2 $(\{x_1 := xx, x_2 := bayab\}, \{x : \{ww \mid w \in \Sigma^+\}, y : \Sigma^+\}), (\{x_1 := aba, x_2 := baaab\}, \{\})$ are substitutions. $(\epsilon, \{\})$ is an empty substitution.

In order to prevent an inappropriate substitution, we define interpretations of terms and applicability of substitutions as follows.

Definition 11 Let $\tau = (\tau', \{x_1 : \mathcal{T}_1, \ldots, x_n : \mathcal{T}_n\})$ be a typed term, where $v(\tau') = \{x_1, \ldots, x_n\}$. Then the *interpretation* $\overline{\tau}$ of τ is $\{\tau'\theta \mid \theta = \{x_1 := \pi_1, \ldots, x_n := \pi_n\},\ \pi_j \in \mathcal{T}_j\}(\subseteq \Sigma^+)$.

Example 3 For the typed terms $(x, \{x : \mathcal{T}\}), (ab, \{\}), (xax, \{x : \mathcal{T}\})$, their interpretations are $\overline{(x, \{x : \mathcal{T}\})} = \mathcal{T}$, $\overline{(ab, \{\})} = \{ab\}$, $\overline{(xax, \{x : \mathcal{T}\})} = \{waw \mid w \in \mathcal{T}\}$, respectively.

Definition 12 Let $E = (E', \{x_1 : \mathcal{T}_1, \ldots, x_n : \mathcal{T}_n\})$ be a typed term(typed definite clause) and $\theta = (\{x_{i_1} := \tau'_1, \ldots, x_{i_k} := \tau'_k\}, C)$ be a substitution, where E' is a term(definite clause), $\{x_1, \ldots, x_n\} = v(E')$, $\tau_i = (\tau'_i, C(\tau'_i))$ is a typed term, $C = C(\tau'_1) \cup \cdots \cup C(\tau'_k)$. θ is *applicable* to E iff the two following conditions hold:

(1) For any j $(1 \leq j \leq k)$, $\overline{\tau_j} \subseteq \mathcal{T}_l$ if there exists an x_l $(1 \leq l \leq n)$ such that $x_{i_j} \equiv x_l$.

(2) $(C(E') \setminus \{x_i : \mathcal{T}_i; x_i \in D(\theta),\ 1 \leq i \leq n\}) \cup \bigcup_j C(\tau'_j) (= C(E'\theta))$ is consistent, where $\bigcup_j C(\tau'_j)$ is a union on j $(1 \leq j \leq k)$ such that there exists an x_l $(1 \leq l \leq n)$ such that $x_{i_j} \equiv x_l$

Then the typed term(typed definite clause) $E\theta$ is $(E'\theta, C(E'\theta))$.

In Definition 12, (1) is the condition for checking types, (2) is the condition to guarantee that $E\theta$ is a typed term (typed definite clause). If θ is ground, $\overline{\tau_j} \subseteq \mathcal{T}_l$ of (1) means $\tau_j \in \mathcal{T}_l$, (2) clearly holds.

Let us apply only the substitutions which satisfy the above two conditions to typed tems (typed definite clauses). then we define a typed EFS, which is a natural extension of an EFS as follows.

Definition 13 A *typed EFS* is a triplet (Σ, Π, Γ), where Γ is a finite set of typed definite clauses.

A typed EFS is a natural extension of an EFS. Actually, if all variables in the typed term (typed definite clause) are typed by Σ^+ and all contexts of the substitutions which is used in the proof are $\{z_1 : \Sigma^+, \ldots, z_n : \Sigma^+\}$, then the typed EFS becomes an ordinary EFS.

We define various concepts such as provability, $L(S, p)$, proof figures and variable-boundedness in the same way as in the ordinary EFS's. Hence the following theorem holds.

Theorem 2 Let $S = (\Sigma, \Pi, \Gamma)$ be a typed variable-bounded EFS, and A be a ground atom of S. If $\Gamma \vdash A \leftarrow$, then there exists a normalized proof figure of $A \leftarrow$.

We also define the derivation procedure, $SS(S)$ and $PS(S)$ in the same way as in the ordinary EFS's. Hence the following lemma and theorems hold in typed EFS's.

Lemma 1 Let $\alpha and \beta$ be a pair of typed terms or typed atoms. If one of them is ground, then every unifier of α and β is ground and the set of all unifiers $U(\alpha, \beta)$ is finite and computable.

The following theorem holds from Lemma 1.

Theorem 3 Let S be a typed variable-bounded EFS and G be a ground goal. Then every resolvent of G is ground, and the set of all the resolvents of G is finite and computable.

We can show the following theorem by an induction on the length of normalized proof figures and the length of refutations.

Theorem 4 Let S be a typed variable-bounded EFS. Then the following equation holds:

$$SS(S) = PS(S).$$

Proof. First we show $PS(S) \subseteq SS(S)$. Suppose $PS(S) \ni A \leftarrow$, i.e., $\Gamma \vdash A \leftarrow$. From Theorem 2, there exists a normalized proof figure of $A \leftarrow$. We prove it by an induction on the length of normalized proofs. If $n = 1$, then $\Gamma \ni A \leftarrow$. We can construct a refutation of length 1, that is, $A \in SS(S)$. If $n = 2$, then there are two cases:

a) There exist an axiom $A' \leftarrow$ of S and a substitution θ which satisfy $A \equiv A'\theta$ and have the proof figure Fig. 9.

b) There exist clauses $A \leftarrow B_1$ and $B_1 \leftarrow$ which are ground instances of axioms of S and have the proof figure Fig. 10.

$$\frac{A' \leftarrow}{A \leftarrow}\ (\mathrm{SUB}) \qquad\qquad \frac{A \leftarrow B_1 \quad B_1 \leftarrow}{A \leftarrow}\ (\mathrm{MP})$$

Fig. 9 Fig. 10

In each case, we can construct a refutation of length 1 or 2. Hence $A \in SS(S)$.

Suppose the hypothesis of induction holds if $n \leq k$ $(k \geq 2)$. Consider $n = k + 1$. A normalized proof figure is of the form of Fig. 11, where $B_1, \ldots, B_m$ are ground and $A \leftarrow B'_1, \ldots, B'_m$ is an axiom of S.

$$\cfrac{\cfrac{\cfrac{\cfrac{A' \leftarrow B'_1, \ldots, B'_m \quad \vdots}{A \leftarrow B_1, \ldots, B_m} \quad B_m \leftarrow \quad \vdots}{A \leftarrow B_1, \ldots, B_{m-1}} \quad B_{m-1} \leftarrow}{A \leftarrow B_1, \ldots, B_{m-2}} \quad \vdots}{A \leftarrow B_1} \qquad \begin{matrix}\vdots \\ B_1 \leftarrow\end{matrix}$$
$$A \leftarrow$$

Fig. 11

$length(\Pi_{B_i \leftarrow}) < length(\Pi_{A \leftarrow}) = k+1$, and hence $B_i \in SS(S)$ from the hypothesis of induction. We can get $\leftarrow B_1, \ldots, B_m$ as a resolvent of $\leftarrow A$ and $A' \leftarrow B'_1, \ldots, B'_m$. Therefore we can construct a refutation from $\leftarrow A$, that is, $A \in SS(S)$. Thus we have $PS(S) \subseteq SS(S)$.

Now we show $SS(S) \subseteq PS(S)$ by an induction of length of refutations. If $n = 1$, then there are two cases:

a) $A \leftarrow$ itself is an axiom of S.

b) There exist an axiom $A' \leftarrow$ of S and a substitution θ such that $A \equiv A'\theta$.

In each case, $\Gamma \vdash A$, that is, $A \in PS(S)$.

Suppose the hypothesis of induction holds for $n \leq k$ $(k \geq 2)$. Consider $n = k + 1$. Then there exist an axiom of S, $A' \leftarrow B'_1, \ldots, B'_m$ $(m \geq 1)$ and a substitution θ such that $A \equiv A'\theta$. By using them, we can get a resolvent $\leftarrow B_1, \ldots, B_m$ of $\leftarrow A$ and $A' \leftarrow B'_1, \ldots, B'_m$, where $B_i \equiv B'_i\theta$. Clearly there exist a refutation from $\leftarrow B_i$ which is ground for any i, whose length is less than $k + 1$. From the hypothesis of induction, $\Gamma \vdash B_i \leftarrow$. Hence $\Gamma \vdash A' \leftarrow B'_1, \ldots, B'_m$, $\Gamma \vdash A \leftarrow B_1, \ldots, B_m, \ldots, \Gamma \vdash A \leftarrow$, that is, $A \in PS(S)$. Thus we have $SS(S) \subseteq PS(S)$.

Therefore we have $SS(S) = PS(S)$. □

Now we discuss types defined by EFS, which satisfy the condition (*) on decidability. As a special case of these types, we define a type called $Terms$ and we will illustrate the first order logic in the framework of the typed EFS which uses $Terms$.

Definition 14 Let $\mathcal{T}(\subseteq \Sigma^+)$ be a type. If there exists an EFS S and a predicate symbol p such that $L(S,p) = \mathcal{T}$, $\mathcal{T}$ is called a *type definable by EFS*.

The advantage of these types is that we can use the refutation procedure for type checking. In order to check whether $\pi \in \mathcal{T}$ or not, where $\mathcal{T}$ is defined by a certain EFS S, it suffices to check the derivation from the goal $\leftarrow p(\pi)$.

The *append* program is often used in Prolog textbooks [1, 3]. We can describe it in a typed EFS.

Example 4 Let $S = (\Sigma, \Pi, \Gamma)$ be a typed EFS, where $\Sigma = \{c_s, n_l, a, b, \ldots X, Y, \ldots, \langle, \rangle, ;\}$, $\Pi = \{ap\}$ and

$$\Gamma = \left\{ \begin{array}{l} (ap(n_l, x, x) \leftarrow, \{x : Terms\}), \\ (ap(c_s\langle x; y\rangle, z, c_s\langle x; w\rangle) \leftarrow ap(y, z, w), \\ \{x : Terms, y : Terms, z : Terms, w : Terms\}) \end{array} \right\}.$$

$Terms$ is a type definable by an ordinary EFS $S' = (\Sigma, \{tm\}, \Gamma')$ with

$$\Gamma' = \left\{ \begin{array}{l} tm(n_l) \leftarrow, \ tm(a) \leftarrow, \ \cdots, \\ tm(X) \leftarrow, \cdots, \\ tm(c_s\langle x; y\rangle) \leftarrow tm(x), tm(y) \end{array} \right\}$$

Variables and auxiliary symbols such as a pair of parenthases $(,)$ and comma $','$ which are used to construct terms of first order logic are represented by $X, Y, \ldots, \langle, \rangle, ;$, respectively, to avoid a confusion. A refutation from a goal $\leftarrow ap(x, y, c_s\langle a; c_s\langle b; n_l\rangle\rangle)$ is depicted in Fig. 12. There may be a couple of unifiers that we can use in every step of the derivation procedure, but we can decide which unifiers to use by checking the applicability of the substitution.

The above example shows that the resolution principle of the first order logic corresponds to the resolution principle and type-checking of typed EFS. All the axioms of S' are regular [2] so that we can see $Terms$, the set of all terms of the first order logic, is a context-free language.

5 Concluding Remarks

In this paper, we have formalized typed EFS's as natural extensions of EFS's by introducing types into EFS's. Then we have proved the completeness of refutation which is the basic property as a logic programming language of typed EFS's, and illustrated the first order logic in the framework of the typed EFS to show the power of typed EFS's. We have also defined proof figures for EFS's to give the concise proof of Theorem 4 and shown there exist standard proof figures in proof figures of ground atoms which give the semantics of EFS's.

As future works, there are problems related to machine learning, especially to language learning. We believe that the introduction of types into EFS's contribute to improve efficiency in machine learning and also to make it easy to apply algorithmic/computational learning theory to practical problems.

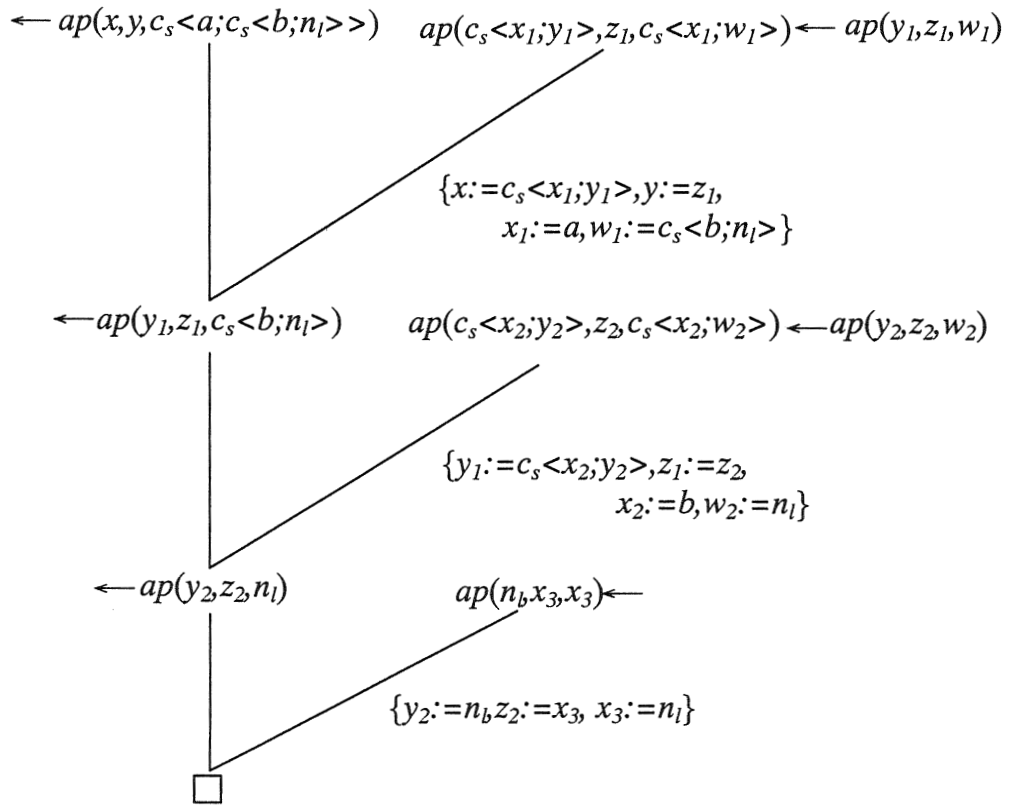

Fig. 12:A refutation from $\leftarrow ap(x, y, c_s\langle a; c_s\langle b; n_l\rangle\rangle)$

References

[1] Arikawa, S., Haraguchi, M., *Predicate Logic and Logic Programming* (in Japanese), Ohmsha, 1988.

[2] Arikawa, S., Shinohara, T., Yamamoto, A., Learning elementary formal systems, *Theoretical Computer Science* **95**, 97–113, 1992.

[3] Goto, S., *Introduction to PROLOG* (in Japanese), Science-sha, 1984.

[4] Hayashi, S., *Mathematical Logic* (in Japanese), Corona-sha, 1989.

[5] Kobayashi, S., Program extraction from constructive proofs (in Japanese), *Computer Software* **7**, 319–334, 1990.

[6] Lloyd, J. W., *Foundations of Logic Programing* (Second Extended Edition), Springer-Verlag, 1987.

[7] Martin-Löf, P., *Constructive mathematics and computer programming ; Mathematical Logic and Programming Languages*, Prentice-Hall International, 1985.

[8] Smullyan, R. M., *Theory of Formal Systems*, Princeton University Press, 1961.

[9] Tatsuta, M., Type theory I, II, III, IV (in Japanese), *Computer Software* **8**, 1991.