Consistent Annotations for Scope Rules

MASATO TAKEICHI*

This paper gives several consistent specifications for the scope rules of block-structured languages. They are considered as annotations for the scope rules defined in existing languages. Annotations will help the user to understand the scope rules and give unambiguous definitions of the rules to the implementor. Our annotations aim at providing a concise description for the user and deriving efficient algorithms for implementing the scope analysis routines in compilers.

In Section 1, the motivation of our research is presented. Section 2 introduces a simple model for the scope and gives several definitions including one for the traditional scope rules. In Section 3, annotations using simple operators are presented and their consistency is proved. Section 4 gives another kind of annotation which is proved to be consistent with the previous ones. This is useful in implementing scope analysis routines. In Section 5, a new kind of scope rules is introduced and annotations developed in previous sections are modified. Section 6 is devoted to discussing the scopes not treated in this paper. And the concluding remarks are given in Section 7.

1. Introduction

Both users and implementors of a programming language need a description which is comprehensible and unambiguous. Notation such as Backus-Naur form (BNF) has been widely used to specify the syntax, whereas notation for describing the semantics has been developed much slower. Natural languages have been generally used to describe the semantics with the syntax specified by BNF. Recently, several kinds of formal definition of the semantics have been proposed [1, 2]. Such definitions sometimes require the user of the language to follow lengthy formulas for understanding the grammar. The complexity of formal definition for existing language is the largest problem in describing the semantics. Although some part of the problem is due to the logical and semantical irregularities of current languages [3], it would be difficult to develop a practical language concisely defined by some formal method proposed today. If such formal definition were given for providing the common standard of the language, additional informal description would be preferable for most users.

Many languages evolved during the period of progress in the formal semantics have been originally defined by traditional and informal method [4, 5, 6, 7]. Among several problems found in such informal description, the scope rules of identifiers have been argued again and again. Although there seems no difference in interpretation of addition of integers, the scope rules are likely to differ in several implementations. Correct understanding of the scope rule is most important for the user to write modular programs. Therefore, the scope rule is necessarily described in some formal way avoiding voluminous definition of the semantics of the whole language. A quick and convenient method is to annotate the scope rules by means of mathematical notation.

This paper gives several consistent annotations for the scope rules, which include so-called closed scope in addition to traditional open scope. Although one may take any of these annotations as definition of the scope rule and others as different interpretations for the rule, we will give a definition in the next section and consistent annotations thereafter. These annotations aim also at providing convenient algorithms for implementation.

2. Simple Model for Scope Rules

In the traditional scope rules of Algol-like languages such as Pascal, an identifier is declared in a block and is automatically inherited in all constituent blocks unless the identifier is redeclared in inner blocks. The definition of the identifier is not visible outside the block in which the declaration occurs. Such automatic inheritance with restricted visibility of identifiers is called an *open scope*.

Formulas in predicate calculus or lambda calculus obey similar scope rules. In a lambda expression $\lambda x.A$, for example, variable x is bound to A and it is inherited in all subexpressions in A. Such bound variable corresponds to the declared identifier in programming languages.

In order to describe the scope rules formally, we will define a class of formulas which models the block structure of programming languages. Although this simplifications is not essential, it is useful to present our annatations concisely.

Definition (Class of formulas F)
Symbol of F
S1) Variables $x, y, z, \cdots$
S2) Scope control symbols $\langle$, | , $\rangle$, [,] and comma (,)

*Department of Computer Science, The University of Electro-Communications.

Formula of F

F1) Variable x is a formula in F.

F2) If x is a variable and $A_1, \cdots, A_n$ $(n \geqq 1)$ are formulas in F,

$\langle x|A_1, \cdots, A_n\rangle$ is a formula in F.

F3) Only the formulas specified by F1) and F2) are formulas in F.

Here, we exclude symbols like the logical connectives in logical formulas. If such symbols were included with the restriction that they do not affect the scope of variables, the following discussion could be done similarly. Symbols [and] will be used in Sec. 5 for extending the class of formulas so that it reflects new kinds of scope rules.

Abbreviation (List of formulas)

A) If $A_1, \cdots, A_n$ $(n \geqq 1)$ are formulas in F, a list

$$A_1, \cdots, A_n$$

is abbreviated as

$$\bar{A}.$$

Using A), formula by Definition F2) is written as $\langle x|\bar{A}\rangle$. Formula $\langle x|\bar{A}\rangle$ is a model of a block which contains a declaration for identifier x and block body A, e.g., a fragment of Algol-60 program:

```
BEGIN REAL x;
    A
END.
```

Note that the above definition of formula reflects nested blocks in programming languages.

We will allocate unique *location* to each symbol in a formula. For simplicity the location is assumed to be the ordinal number from the left of the formula. The location is represented as superscript of the symbol. Locations assigned to scope control symbols are usually omitted. Every occurrence of variable x^i in a formula of the form $\langle x^i|\bar{A}\rangle$ is called a *defining occurrence* of x at i. Other occurrences of variables are called *applied occurrences*. The scope rules defined in programming language state that each applied occurrence of an identifier should be identified with which defining occurrence of the same identifier. In order to represent such identification relation, we will assign an *identification tag* to each applied occurrence of variable in a formula. The identification tag is attached to the variable as a subscript. Hereafter, when we refer to the formula, it is assumed that each symbol has the location and each applied occurrence of a variable has an identification tag.

Definition (Equality of formulas)

Formulas A_1 and A_2 are equal if

1) symbols in formulas A_1 and A_2 located at the same location are the same, and
2) applied occurrences of variables in formulas A_1 and A_2 at the same location have the same identification tag.

Equality relation between A_1 and A_2 is written as $A_1 = A_2$.

Definition (Constituent of formula)

If a formula B is of the form $\langle x^i|A_1, \cdots, A_n\rangle$ then $A_1, \cdots, A_n$ are *immediate con tituents* of B.

A formula A is a *constituent* of B, either

1) if A is an immediate constituent of B, or
2) if A is an immediate constituent of a constituent of B.

A formula A *is embraced by* B if A is a constituent of B. In this case, B *embraces* A.

Definition (Open scope rule)

If every applied occurrence of variable x in formula A has the identification tag j, it is the location of the defining occurrence of the same variable x located at the smallest formula of the form $\langle x^j|\bar{A}\rangle$ embracing that applied occurrence. If no such defining occurrence, $j=0$.

This definition of the open scope rule is commonly used in interpreting the scope rules of Algol-60 [8].

Definition (Raw and refined formulas)

A formula in which every applied occurrence has identification tag 0 is called a *raw formula*.

A formula in which every applied occurrence has an identification tag assigned according to the scope rule defined as above is called a *refined formula*.

3. Operational Annotation

Our next approach to describe the scope rules is based on operations which are applied to any formula to yield the refined formula. Such transformation preserves the structure of the original formula. This property will not be stated explicitly hereafter. Our approach here comes from the notation used in the definition of Algol-N [9].

We will define two operations r and s to specify the transformation. These operations are defined according to the structure of the formula.

Definition (r-s)

r1) $r(x_j^i) = x_j^i$

r2) $r(\langle x^i|\bar{A}\rangle) = \langle x^i|r(s(\bar{A}; i/x))\rangle$

s1) $s(x_j^i; k/z) = x_k^i$ if $x = z$,
x_j^i otherwise

s2) $s(\langle x^i|\bar{A}\rangle; k/z) = \langle x^i|s(\bar{A}; k/z)\rangle$

where

$$r(\bar{A}) = r(A_1), \cdots, r(A_n)$$

and

$$s(\bar{A}; k/z) = s(A_1; k/z), \cdots, s(A_n; k/z)$$

when $\bar{A} = A_1, \cdots, A_n$.

Operation r is applied recursively to constituent formulas and operation s gives identification tags to variables. Rule r2) represents the effect of the declaration and Rule s2) does its inheritance to inner blocks.

We can state the open scope rules in terms of these operations.

Annotation (r-s)

If A is a raw formula, the identification tag assigned to each applied occurrence of a variable in $r(A)$ is the location of the defining occurrence with which it should be identified.

If an applied occurrence of a variable in $r(A)$ has identification tag 0, there is no defining occurrence for that occurrence at all.

Example 1.

$$A=\langle x^2|\langle y^5|\langle x^8|x_0^{10}, y_0^{12}\rangle, x_0^{15}\rangle\rangle$$
$$\begin{aligned}r(A)&=\langle x^2|r(s(\langle y^5|\langle x^8|x_0^{10}, y_0^{12}\rangle, x_0^{15}\rangle; 2/x))\rangle\\&=\langle x^2|r(\langle y^5|\langle x^8|x_2^{10}, y_0^{12}\rangle, x_2^{15}\rangle)\rangle\\&=\langle x^2|\langle y^5|r(s(\langle x^8|x_2^{10}, y_0^{12}\rangle; 5/y)), x_2^{15}\rangle\rangle\\&=\langle x^2|\langle y^5|r(\langle x^8|x_2^{10}, y_5^{12}\rangle), x_2^{15}\rangle\rangle\\&=\langle x^2|\langle y^5|\langle x^8|r(s(x_2^{10}, y_5^{12}; 8/x))\rangle, x_2^{15}\rangle\rangle\\&=\langle x^2|\langle y^5|\langle x^8|r(x_8^{10}, y_5^{12})\rangle, x_2^{15}\rangle\rangle\\&=\langle x^2|\langle y^5|\langle x^8|x_8^{10}, y_5^{12}\rangle, x_2^{15}\rangle\rangle\end{aligned}$$

Note that the identification tag for x at location 10 has been changed twice.

Lemma 1.

If formula A is of the form $\langle x^k|\bar{B}\rangle$, then every applied occurrence of variable x in $r(A)$ has identification tag not less than k.

Proof is immediate from Definition (r-s) above. Recall that the structure of formulas are preserved by operations r and s.

Theorem 1.

If A is a raw formula, $r(A)$ is the refined formula.

Proof.

Let x_j^i be an applied occurrence of variable x in $r(A)$. We will show that j is the correct identification tag of the refined formula.

Case 1. $j=0$

In this case, there is no defining occurrence of x, $\langle x^k|\bar{B}\rangle$ where $\bar{B}$ embraces x_j^i. If such k ever exists, $j\geqq k>0$ follows from Lemma 1. This contradicts $j=0$.

Case 2. $j>0$

Assume that formula $\langle x^k|\bar{B}\rangle$ embraces x_j^i.

Lemma 1 states that $j\geqq k$. This holds for any formula of the form $\langle x^k|\bar{B}\rangle$. It is obvious that j really is equal to $k0$ if $\langle x^{k0}|\bar{B}\rangle$ happends to be the smallest one, and $k0$ is the largest such k. That is, $j=k0$.

Annotation (r-s) for the open scope rule is straightforward from the informal specification described in the first paragraph in Sec. 2. It is, however, inefficient if we apply the operation to implementation of scope analysis routine. More sophisticated annotation using similar operations would be convenient for implementation.

Definition (r'-s')

r'1) $r'(x_j^i)=x_j^i$

r'2) $r'(\langle x^i|\bar{A}\rangle)=\langle x^i|r'(s'(\bar{A}; i/x))\rangle$

s'1) $s'(x_j^i; k/z)=x_k^i$ if $x=z$,
x_j^i otherwise

s'2) $s'(\langle x^i|\bar{A}\rangle; k/z)=\langle x^i|\bar{A}\rangle$ if $x=z$,
$\langle x^i|s'(\bar{A}; k/z)\rangle$ otherwise

where

$$r'(\bar{A})=r'(A_1), \cdots, r'(A_n)$$

and

$$s'(\bar{A}; k/z)=s'(A_1; k/z), \cdots, s'(A_n; k/z)$$

when $\bar{A}=A_1, \cdots, A_n$.

Here, we can see the restricted inheritance of declarations in rules r'2) and s'2).

Annotation (r'-s')

Annotation using r' and s' is obtained by simply replacing r in Annotation (r-s) by r'.

Example 2.

For the same formula A as Example 1,

$$\begin{aligned}r'(A)&=\langle x^2|r'(s'(\langle y^5|\langle x^8|x_0^{10}, y_0^{12}\rangle, x_0^{15}\rangle; 2/x))\rangle\\&=\langle x^2|r'(\langle y^5|\langle x^8|x_0^{10}, y_0^{12}\rangle, x_2^{15}\rangle)\rangle\\&=\langle x^2|\langle y^5|r'(s'(\langle x^8|x_0^{10}, y_0^{12}\rangle; 5/y)), x_2^{15}\rangle\rangle\\&=\langle x^2|\langle y^5|r'(\langle x^8|x_0^{10}, y_5^{12}\rangle), x_2^{15}\rangle\rangle\\&=\langle x^2|\langle y^5|\langle x^8|r'(s'(x_0^{10}, y_5^{12}; 8/x))\rangle, x_2^{15}\rangle\rangle\\&=\langle x^2|\langle y^5|\langle x^8|x_8^{10}, y_5^{12}\rangle, x_2^{15}\rangle\rangle\end{aligned}$$

Now, we want to establish the consistency of Annotations (r-s) and (r'-s').

Lemma 2.

For any formula A, any variable z, and any location k,

$$r'(s'(A; k/z))=s'(r'(A); k/z).$$

That is, r' and s' are commutative.

Proof.

By induction on the nesting depth d of $\langle\ |\ \rangle$ construct.

Case 1. $d=0$

Assume $A=x_j^i$.

$$\begin{aligned}r'(s'(x_j^i; k/z))&=r'(x_k^i) \text{ if } x=z,\ r'(x_j^i) \text{ otherwise}\\&=x_k^i \text{ if } x=z,\ x_j^i \text{ otherwise}\end{aligned}$$

$$\begin{aligned}s'(r'(x_j^i); k/z)&=s'(x_j^i; k/z)\\&=x_k^i \text{ if } x=z,\ x_j^i \text{ otherwise}\end{aligned}$$

Thus, r' and s' are commutative.

Case 2. Assume that the maximum depth of $\langle\ |\ \rangle$ in formulas $A_1, \cdots, A_n$ $(n \geqq 1)$ is $d \geqq 0$, and r' and s' are commutative for these formulas. Then we will show that these operations are commutative for formula $\langle x^i|\bar{A}\rangle$ of depth $d+1$, where $\bar{A}=A_1, \cdots, A_n$.

$r'(s'(\langle x^i|\bar{A}\rangle; k/z))$
$=r'(\langle x^i|\bar{A}\rangle)$ if $x=z$, $r'(\langle x^i|s'(\bar{A}; k/z)\rangle)$ otherwise
$=\langle x^i|r'(s'(\bar{A}; i/x))\rangle$ if $x=z$,
$\langle x^i|r'(s'(s'(\bar{A}; k/z); i/x))\rangle$ otherwise

$s'(r'(\langle x^i|\bar{A}\rangle); k/z)=s'(\langle x^i|r'(s'(\bar{A}; i/x))\rangle; k/z)$
$=\langle x^i|r'(s'(\bar{A}; i/x))\rangle$ if $x=z$,
$\langle x^i|s'(r'(s'(\bar{A}; i/x)); k/z)\rangle$ otherwise

By the induction hypothesis, the last formula becomes

$\langle x^i|s'(s'(r'(\bar{A}); i/x); k/z)\rangle$ if $x \neq z$
$=\langle x^i|s'(s'(r'(\bar{A}); k/z); i/x)\rangle$ if $x \neq z$

from a basic property of s'.
That is, r' and s' are commutative.

Lemma 3.
For any formula A, any variable z, and any location k,

$$r'(s'(A; k/z))=r'(s(A; k/z)).$$

Proof.
By induction on the nesting depth d of $\langle\ |\ \rangle$ construct.
Case 1. $d=0$
Assume $A=x_j^i$. Proof is immediate.
Case 2.
Assume that the equality holds for all the A_p's in $\bar{A}$.

$r'(s(\langle x^i|\bar{A}\rangle; k/z))$
$=r'(\langle x^i|s(\bar{A}; k/z)\rangle)$
$=\langle x^i|r'(s'(s(\bar{A}; k/z); i/x))\rangle$
$=\langle x^i|s'(r'(s(\bar{A}; k/z)); i/x)\rangle$ from Lemma 2.
$=\langle x^i|s'(r'(s'(\bar{A}; k/z)); i/x)\rangle$ from hypothesis
$=\langle x^i|r'(s'(s'(\bar{A}; k/z); i/x))\rangle$ from Lemma 2.
$=r'(s'(\langle x^i|\bar{A}\rangle; k/z))$

Lemma 4.
For any formula A, any variable z, and any location k,

$$r'(s'(A; k/z))=r(s(A; k/z)).$$

Proof proceeds similarly as above using Lemmas 2 and 3.

Theorem 2.
For any formula A,

$$r'(A)=r(A).$$

That is, Annotations (r-s) and (r'-s') are consistent.

Proof is immediate from Lemma 4.

4. Annotation Using Environment

In the previous section, two annotations using primitive operations have been given, where the substitution operators s and s' play a role in assigning identification tags to applied occurrences. When these operations are applied to a list of formulas, some constituent formulas may remain unchanged which are possibly changed by later substitution. This is because the substitution operates one variable at a time and the formulas are necessarily scanned many times. Such operations are inconvenient for efficient implementation. If we have all the substitution parameters k/z for all the variables concerned, we can assign proper identification tags to all applied occurrences through a single scan of the list of formulas. In order to realize such algorithm, we will introduce a kind of representation for *environment* of the scope.

Definition (Environment)
Class *Env* of Environment Functions
Env1) Φ_0 is in *Env*.
Env2) If Φ is in *Env*, z is a variable, and k is a location, $\Phi\{k/z\}$ is in *Env*.
Env3) Only functions defined by Env1) and Env2) are in *Env*.
Every Φ in *Env* is a function
Φ: Variable$\rightarrow$Location
defined by
Φ1) $\Phi_0(x)=0$ for any variable x.
Φ2) If the function is of the form $\Phi\{k/z\}$,

$$\Phi\{k/z\}(x)=k \text{ if } x=z,$$
$$\Phi(x) \text{ otherwise}$$

Environment function Φ is a representation of the symbol table which is commonly used in many implementations of the scope analysis routine. We now define an operation to refine formulas with the use of environment.

Definition (R)
R1) $R(x_j^i; \Phi)=x_{\Phi(x)}^i$
R2) $R(\langle x^i|\bar{A}\rangle; \Phi)=\langle x^i|R(\bar{A}; \Phi\{i/x\})\rangle$
where
$R(\bar{A}; \Phi)=R(A_1; \Phi), \cdots, R(A_n; \Phi)$
when
$\bar{A}=A_1, \cdots, A_n$.

Annotation $(R-\Phi)$
If A is a raw formula, the identification tag assigned to each applied occurrence of variable in $R(A; \Phi_0)$ is the location of the defining occurrence with which it should be identified.

Example 3.

Given the same formula A as Examples 1 and 2.

$$R(A;\Phi_0)=\langle x^2|R(\langle y^5|\langle x^8|x_0^{10},y_0^{12}\rangle,x_0^{15}\rangle;\Phi_0\{2/x\})\rangle$$
$$=\langle x^2|\langle y^5|R(\langle x^8|x_0^{10},y_0^{12}\rangle;\Phi_0\{2/x\}\{5/y\},R(x_0^{15};\Phi_0\{2/x\}\{5/y\})\rangle\rangle$$
$$=\langle x^2|\langle y^5|\langle x^8|R(x_0^{10},y_0^{12};\Phi_0\{2/x\}\{5/y\}\{8/x\}),x_2^{15}\rangle\rangle$$
$$=\langle x^2|\langle y^5|x^8|\langle x_8^{10},y_5^{12}\rangle,x_2^{15}\rangle\rangle$$

Lemma 5.

For any formula A, any environment Φ, any variable z, and any location k,

$$R(A;\Phi\{k/z\})=s'(R(A;\Phi);k/z).$$

Proof.

Case 1. Assume $A=x_j^i$.

$$R(x_j^i;\Phi\{k/z\})=x^i_{\Phi\{k/z\}(x)}$$
$$=x_k^i \text{ if } x=z,$$
$$x^i_{\Phi(x)} \text{ otherwise}$$
$$s'(R(x_j^i;\Phi);k/z)=s'(x^i_{\Phi(x)};k/z)$$
$$=x_k^i \text{ if } x=z,$$
$$x^i_{\Phi(x)} \text{ otherwise}$$

That is, $R(x_j^i;\Phi\{k/z\})=s'(R(x_j^i;\Phi);k/z)$.

Case 2. Assume that

$$R(A_p;\Phi\{k/z\})=s'(R(A_p;\Phi);k/z)$$

for $p=1,\cdots,n$.

We will prove that for $\bar{A}=A_1,\cdots,A_n$,

$$R(\langle x^i|\bar{A}\rangle;\Phi\{k/z\})=s'(R(\langle x^i|\bar{A}\rangle;\Phi);k/z)$$

holds.

$$R(\langle x^i|\bar{A}\rangle;\Phi\{k/z\})$$
$$=\langle x^i|R(\bar{A};\Phi\{k/z\}\{i/x\})\rangle$$
$$=\langle x^i|s'(R(\bar{A};\Phi\{k/z\});i/x)\rangle \text{ from hypothesis}$$
$$=\langle x^i|s'(s'(R(\bar{A};\Phi);k/z);i/x)\rangle \text{ as above}$$

$$s'(R(\langle x^i|\bar{A}\rangle;\Phi);k/z)$$
$$=s'(\langle x^i|R(\bar{A};\Phi\{i/x\})\rangle;k/z)$$
$$=s'(\langle x^i|s'(R(\bar{A};\Phi);i/x)\rangle;k/z) \text{ from hypothesis}$$
$$=\langle x^i|s'(R(\bar{A};\Phi);i/x\rangle \text{ if } x=z,$$
$$\langle x^i|s'(s'(R(\bar{A};\Phi);i/x);k/z)\rangle \text{ otherwise}$$

In general,

$$s'(s'(A;k/z);i/x)=s'(A;i/x) \text{ if } x=z$$

and

$$s'(s'(A;k/z);i/x)=s'(s'(A;i/x);k/z) \text{ if } x\neq z$$

hold.

Thus,

$$R(\langle x^i|\bar{A}\rangle;\Phi\{k/z\})=s'(R(\langle x^i|\bar{A}\rangle;\Phi);k/z).$$

Lemma 6.

If $R(A_p;\Phi)=r(A_p)$ for $p=1,\cdots,n$, then

$$R(\langle x^i|\bar{A}\rangle;\Phi)=r(\langle x^i|\bar{A}\rangle)$$

where $\bar{A}=A_1,\cdots,A_n$.

Proof.

$$R(\langle x^i|\bar{A}\rangle;\Phi)=\langle x^i|R(\bar{A};\Phi\{i/x\})\rangle$$
$$=\langle x^i|s'(R(\bar{A};\Phi);i/x)\rangle \text{ from Lemma 5}$$
$$=\langle x^i|s'(r(\bar{A});i/x)\rangle \text{ from assumption}$$
$$=\langle x^i|s'(r'(\bar{A});i/x)\rangle \text{ from Theorem 2}$$
$$=\langle x^i|r'(s'(\bar{A};i/x))\rangle \text{ from Lemma 2}$$
$$=r'(\langle x^i|\bar{A}\rangle) \text{ from Definition r'2)}$$
$$=r(\langle x^i|\bar{A}\rangle) \text{ from Theorem 2}$$

Theorem 3.

If A is a raw formula,

$$R(A;\Phi_0)=r(A).$$

That is, Annotations $(R-\Phi)$ and (r-s) are consistent.

Proof is immediate from Lemma 6 and Definition Φ1).

5. Annotation for Closed Scope

The scope rules of Algol-60 can be specified by one of the annotations described so far. The same annotations are applicable to specify large parts of the scope rules in Pascal [4]. That is, the scopes dependent solely on declarations and definitions in blocks have the same property as those of Algol-60.

However, a new kind of scopes is observed in modern languages such as Modula-2[5], Euclid[6], and Ada[7], which are considered as successors of Pascal. In these languages, the programmer can specify explicitly which identifiers are or are not inherited by inner blocks and which ones are visible outside the block where they are declared. Such kind of scope is called a *closed scope* in contrast to the open scope. In Modula-2 and some other languages, explicit inheritance of identifiers from outside of the block is specified by an *import* list, and explicit widening of identifiers declared or exported in the block is specified by an *export* list. Next program segment illustrates import and export lists in Modula-2 modules, which are controlled by the closed scope rule.

```
MODULE M;
VAR x, w:···;
MODULE N;
  IMPORT x;
  EXPORT y;
  VAR y, z:···;
  BEGIN···
    {x, y, and z are visible}
  END N;
BEGIN···
  {x, y, and w are visible}
END M;
```

We will extend the class of formulas F so that it contains formulas reflecting the closed scope rule, and describe

consistent annotations for such extended scopes.

We could define the closed scope as we have done for the open scope in Sec. 2. It becomes, however, tedious because the visibility of identifier can be restricted and widened across the block boundaries. We do not define the closed scope in this way and instead give extended annotations based on previous ones in this paper. Although the definition is not given, one may choose any of annotations as definition. In fact, complex rules generally require operational specification.

Definition (Class of formulas F——extended version)
Definition of F in Sec. 2 is augmented by the next rule.

F2′) If i, e, and x are variables, and $A_1, \cdots, A_n$ ($n \geqq 1$) are formulas in F,

$$[i^a_{a'}|e^b_{b'}|x^c|\bar{A}]$$

is a formula in F.

Locations and identification tags are assigned to new formulas as before. Variables i and e are also considered as applied occurrences and have identification tags. The raw formula and the refined formula in the new class of formulas are straightforward extension of old ones.

Definition (Exported variables)
Function E over formulas giving a location-variable pair is defined as follows:

E1) If A is of the form $x^k_{k'}$, $E(A)=\phi$
E2) If A is of the form $\langle x^c|\bar{B}\rangle$, $E(A)=\phi$
E3) If A is of the form $[i^a_{a'}|e^b_{b'}|x^c|\bar{B}]$, $E(A)=b/e$

where ϕ is a distinguished element.
If $\bar{A}=A_1, \cdots, A_n$, a list of b/e pairs

$$E(A_1), \cdots, E(A_n)$$

is abbreviated as $E(\bar{A})$.

We now need to define a class of *legal formulas* which is characterized according to the restriction on the defining, importing and exporting occurrences of variables.

Definition (Legal formula)
A formula A is *legal* if

L1) A is of the form $x^k_{k'}$, or
L2) A is of the form $\langle x^c|\bar{B}\rangle$ and x does not appear in variables of $E(\bar{B})$, or
L3) A is of the form $[i^a_{a'}|e^b_{b'}|x^c|\bar{B}]$, $i \neq e$, $i \neq x$, and both x and i do not appear in variables of $E(\bar{B})$.

Legality of the formula could be tested in the course of refinement by operators. In the following discussions, however, all formulas are assumed to be legal. If one wishes to specify the scope rules including the rule for legality, it can be annotated in two stages; one for legality, and one for scopes.

Definition (r-s extended version)
Rules for [|||] construct are augmented.

r3) $r([i^a_{a'}|e^b_{b'}|x^c|\bar{A}])$
$=[i^a_{a'}|s(e^b_{b'}; c/x, E(\bar{A}))|x^c|r(s(\bar{A}; c/x, a/i, E(\bar{A})))]$

s3) $s([i^a_{a'}|e^b_{b'}|x^c|\bar{A}]; k/z)=[s(i^a_{a'}; k/z)|e^b_{b'}|x^c|\bar{A}]$

where

$$s(A; k_1/z_1, \cdots, k_m/z_m)=s(s(\cdots s(A; k_1/z_1)\cdots); k_m/z_m)$$

and

$$s(A; \phi)=A.$$

Annotation (r-s extended version)
If A is a raw formula, the identification tag assigned to each applied, importing, or defining occurrence of variable in $r(A)$ is the location of the defining, importing, or exporting occurrence with which it should be identified.

Definition (r′-s′ extended version)
Rules for [|||] construct are included.

r′3) $r'([i^a_{a'}|e^b_{b'}|x^c|\bar{A}])$
$=[i^a_{a'}|s'(e^b_{b'}; c/x, E(\bar{A}))|x^c|r'(s'(\bar{A}; c/x, a/i, E(\bar{A})))]$

s′3) $s'([i^a_{a'}|e^b_{b'}|x^c|\bar{A}]; k/z)=[s'(i^a_{a'}; k/z)|e^b_{b'}|x^c|\bar{A}]$

where

$$s'(A; k_1/z_1, \cdots, k_m/z_m)=s'(s'(\cdots s'(A; k_1/z_1); \cdots); k_m/z_m)$$

and

$$s'(A; \phi)=A.$$

Annotation (r′-s′ extended version)
Annotation by r' and s' is obtained by simply replacing r in Annotation (r-s) above by r'.

Note that the augmented rules for r-s and r'-s' operations are very similar. In fact these operations are the same for the closed scope.

Example 4.

$$A=[z^2_0|y^4_0|x^6|[x^9_0|y^{11}_0|y^{13}|x^{15}_0, y^{17}_0], y^{20}_0]$$

$$r(A)=[z^2_0|s(y^4_0; 6/x, 11/y)|x^6|\ r(s([x^9_0|y^{11}_0|y^{13}|x^{15}_0, y^{17}_0], y^{20}_0; 6/x, 2/z, 11/y))]$$
$$=[z^2_0|y^4_{11}|x^6|r([x^9_6|y^{11}_0|y^{13}|x^{15}_0, y^{17}_0]), y^{20}_{11}]$$
$$=[z^2_0|y^4_{11}|x^6|[x^9_6|s(y^{11}_0; 13/y)|y^{13}|\ r(s(x^{15}_0, y^{17}_0; 13/y, 9/x))], y^{20}_{11}]$$
$$=[z^2_0|y^4_{11}|x^6|[x^9_6|y^{11}_{13}|y^{13}|x^{15}_9, y^{17}_{13}], y^{20}_{11}]$$

Definition ($R-\Phi$ extended version)
New rule is augmented for [|||] construct.

R3) $R([i^a_{a'}|e^b_{b'}|x^c|\bar{A}]; \Phi)$
$=[R(i^a_{a'}; \Phi)|R(e^b_{b'}; \Phi\{c/x\}\{E(\bar{A})\})|x^c|\ R(\bar{A}; \Phi\{c/x\}\{a/i\}\{E(\bar{A})\})]$

where $\{E(\bar{A})\}$ denotes

$$\{E(A_1)\}, \cdots, \{E(A_n)\}$$

when $\bar{A}=A_1, \cdots, A_n$.

Annotation ($R-\Phi$ extended version)

If A is a raw formula, the identification tag assinged to each applied, importing, or exporting occurrence of variable in $R(A; \Phi_0)$ is the location of the defining, importing, or exporting occurrence that it should identify.

Example 5.

Given the same formula as Example 4.

$$\begin{aligned} R(A;\Phi_0)=&[R(z_0^2;\Phi_0)|R(y_0^4;\Phi_0\{6/x\}\{11/y\})|x^6| \\ &R([x_0^9|y_0^{11}|y^{13}|x_0^{15}, y_0^{17}], y_0^{20}; \\ &\qquad \Phi_0\{6/x\}\{2/z\}\{11/y\})] \\ =&[z_0^2|y_{11}^4|x^6|[R(x_0^9;\Phi_0\{6/x\}\{2/z\}\{11/y\})| \\ &R(y_0^{11};\Phi_0\{6/x\}\{2/z\}\{11/y\}\{13/y\})| \\ &R(x_0^{15}, y_0^{17};\Phi_0\{6/x\}\{2/z\}\{11/y\}\{9/x\} \\ &\qquad \{13/y\})], y_{11}^{20}] \\ =&[z_0^2|y_{11}^4|x^6|[x_6^9|y_{13}^{11}|y^{13}|x_9^{15}, y_{13}^{17}], y_{11}^{20}] \end{aligned}$$

Theorem 4.

If A is a raw formula,

$$r(A)=r'(A)=R(A;\Phi_0).$$

That is, Annotations (r-s), (r'-s'), and ($R-\Phi$) are all consistent.

Theorem 4 can be proved in the same way as previous lemmas and theorems.

6. Reservation

We have presented annotations for the scope rules in which the scope is related to the syntax. That is, both open and closed scopes are delineated by prescribed syntactic units such as blocks, procedures, and modules. However, Pascal and its successors have a different kind of scopes, which we will call a *remote scope*. The remote scope is associated with records in Pascal, where the field identifier is introduced in type specification of the record and is used through qualification by variable identifier of that type. Since this kind of scope rules depends on the semantics of the language, i.e., types and variables, our annotations for syntactic scopes are not directly applicable. It requires a new concept to formulate such scope rules. We do not discuss scopes of this kind in this paper.

7. Conclusion

We have described the scope rules for the simplified formula which is a model of the block structure of programming languages. Application of our annotations to actual programming language is straightforward. However, as described in the previous section, our annotations are limited to open and closed scopes which are common in modern programming languages. There needs to be other annotations for specific scope rules.

One may formulate the scope rules in general by the use of attribute grammar [10], where the static semantics specifies the scope rules of the language. Description of the static semantics requires other semantic features of the language and is necessarily written in informal way. Denotational formulation is probably the most sophisticated description of the semantics of the whole language. Though as it is, the description is usually very large. Annotating the scope rules is intended to supplement the definition of existing languages by a few additional mathematical notations.

Our method will give a standard interpretation for the rules which are described informally. Moreover, one of the consistent annotations is available for implementors and the other for users. As mentioned previously, annotations using environments are most convenient for implementing the scope analysis routines. Algorithms based on these annotations are applied to actual implementation [11].

References

[1] GORDON, M. J. C. The Denotational Description of Programming Languages, Springer-Verlag (1979).

[2] HOARE, C. A. R. An axiomatic basis for programming. Comm. *ACM* **12** (1967), 576–580.

]3] HOARE, C. A. R. and LAUER, P. E. Consistent and complementary formal theories of the semantics of programming languages. *Acta Informatica* **3** (1974), 135–153.

[4[JENSEN, K. and WIRTH, N. PASCAL User Manual and Report. Springer-Verlag (1974).

[5[WIRTH, N. MODULA-2. *Berichte des Institut fuer Informatik* Nr. 36, ETH Zuerich (1980).

[6] LAMPSON, B. W., HORNING, J. J., LONDON, R. L., MITCHELL, J. G. and POPEK, G. L. Report on the programming language Euclid. *SIGPLAN Notices* **12** (February 1977).

[7[Reference Manual for the Ada Programming Language.United States Department of Defense (1980).

[8] YONEDA, N. and NOSHITA, K. Lectures on ALGOL 60. Kyoritsu-shuppan (1979). (In Japanese)

[9] SIMAUTI, T., et al. ALGOL N. *Commentarii Mathematici Universitatis Sancti Pauli* **21** (1972), 1–72.

[10] KRISTENSEN, B. B., MADSEN, O. L., MOLLER-PEDERSEN, B. and NYGAARD, K. Beta Language Proposal. *Norwegian Computer Center Working Note* No. 5 (1979).

[11] TAKEICHI, M. Name identification for languages with explicit scope control. *Journal Information Processing* **5** (1982) 45–49.

(Received September 24, 1981: revised November 16, 1981)