CafeOBJ言語システムの名前空間

澤田 寿実
(株) 考作舎
toshi.swd@gmail.com

目次

1

CafeOBJ言語の名前空間と名前表示コマンド

CafeOBJ言語における「名前」，すなわち利用者が特定のオブジェクト[1] を名付け，および参照するために使える文字列の構文規則について述べる．

続いて，モジュールの名前空間の表示コマンド， および特定の名前を持ったオブジェクトの表示コマンドについて説明する．

1.1 CafeOBJ言語の名前空間種別

本章ではCafeOBJ言語システムの名前空間の構造について調べる． 名前自身の構文については別途章を改める．

CafeOBJ言語の名前空間には次の2種がある:

1. トップレベル
   モジュール宣言とview宣言で導入された，それぞれモジュールおよびviewの名前の集合からなる．
2. モジュール
   ある特定のモジュールの中で参照可能な名前の集合であり，これは以下のものからなる:
   - そのモジュールで宣言された以下のオブジェクトの名前:
     a) パラメータの名前
     b) サブモジュールの名前
     c) ソートの名前
     d) オペレータの名前
     e) 変数の名前

[1] 本書ではCafeOBJ言語の利用者にとって中核をなすモノ(概念要素)をオブジェクトと呼ぶ． 厳密にはCaefOBJの形式的な構文定義に出現する各構文要素に対応してオブジェクトが存在し得るが，利用者が形式仕様を記述する上で理解しておくべき話とは無縁である． 知らなければならないのは，モジュール，オペレータ，ソート，公理，変数，パラメータ，それとモジュール式の解釈(それと–あれば–それらの属性)である． 従って本書ではこれのみを「オブジェクト」として参照する． オブジェクトの中には「名前」によって参照出来ないものもある．

f) 公理ラベルの名前

- 輸入しているモジュールの名前空間に含まれる名前のうち，変数名を除いた名前
- トップレベルの名前空間に含まれる名前

以下これらの名前空間の構造について詳細を述べる．

1.1.1 トップレベルの名前空間の構造と参照の解決

上で述べた通りトップレベルの名前空間はモジュールの名前とviewの名前からなり， これらは重なりを持つことがあり得る(図 1.1を参照)．

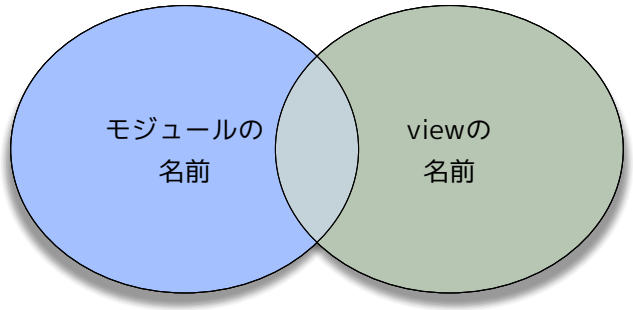

図 1.1: トップレベルの名前空間

すなわち，同一の名前を持つモジュールとviewを宣言する事が可能である． この場合モジュール式(第 1.2.3章を参照)によっては そこに出現する名前がモジュールの名前かviewの名前かで曖昧性が生ずる場合があり得るが， 本書が対象としているCafeOBJインタプリタでは以下のようにして曖昧さを回避する:

1. 指定の名前を持つviewを探す
2. その名前のviewが宣言されていれば，そのviewが指定されているものと解釈する
3. さもなければモジュールの名前が指定されているものと解釈する

1.1.2 モジュールの名前空間の構造と参照の解決

サブモジュールを持たないモジュールの名前空間

まず，輸入しているモジュール(サブモジュール)を持たないモジュールの名前空間の構造を図 1.2に示す:

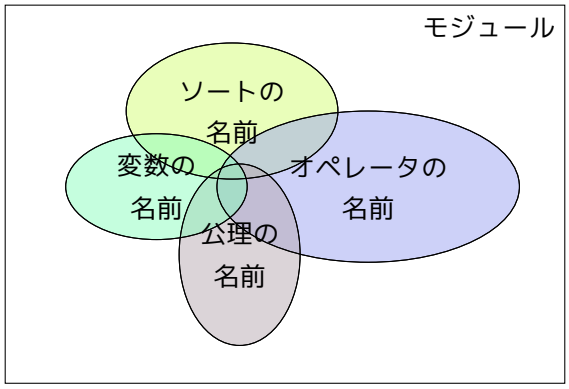

図 1.2: サブモジュールなしのモジュールの名前空間

図から了解されるように，ソート，オペレータ，変数，公理それぞれで互いに同名のものが許される． 通常は名前の出現する文脈(構文)に応じてどの種のオブジェクトについての名前であるかが明確となるため曖昧性が生ずることは無い． 唯一あり得るのは，定数オペレータと変数の名前が同じ場合である．

同名の定数項と変数の参照の解決

まず，他のオペレータの引数に出現している場合 "$f(N)$"を考える． ここで名前 N が定数オペレータ及び変数の名前として宣言されているとする． システムは次のようにして項$f(N)$の解釈を試みる:

1. オペレータfの引数のソートを調べる．たまたまこれがSであったとする．
2. N という名前でかつソートがS以下の定数オペレータを探す．
3. これが見つかった場合Nは定数オペレータの項であると解釈する．
4. 定数オペレータが見つからなかった場合は，N という名前でソートS以下の変数を探す．
5. このような変数が見つかれば N は変数項であると解釈する．
6. 上のいずれでもない場合は構文エラーとする．

これで分かる通り，システムは項が出現する文脈に応じて期待されるソートにより制約をかけ，なおかつオペレータの名前の方を優先して解釈しようとする． 同名の変数と定数オペレータが互いに関係の無いソートのものであれば引数に出現する限りにおいて，曖昧性が生ずることは無い．

次に項 N が単独で出現した場合であるが，これは解釈のための文脈がないため曖昧さを解消する術がない． 従ってこの場合は，項が曖昧であるとした構文エラーとして扱われる．

階層的モジュールの名前空間の構造

あるモジュールが他のモジュールを輸入している時，輸入されているモジュールをサブモジュールと呼ぶ．この時サブモジュールの名前空間に含まれる名前は， **変数の名前を除いて**[2] 輸入しているモ

[2] すなわち，輸入しているモジュールで宣言されている変数を参照することは出来ない．

ジュールの名前空間の一部となる(図 1.3を参照)． 図ではモジュールAがモジュールBを，モジュールBがモジュールCを輸入している場合が図示されている[3]．

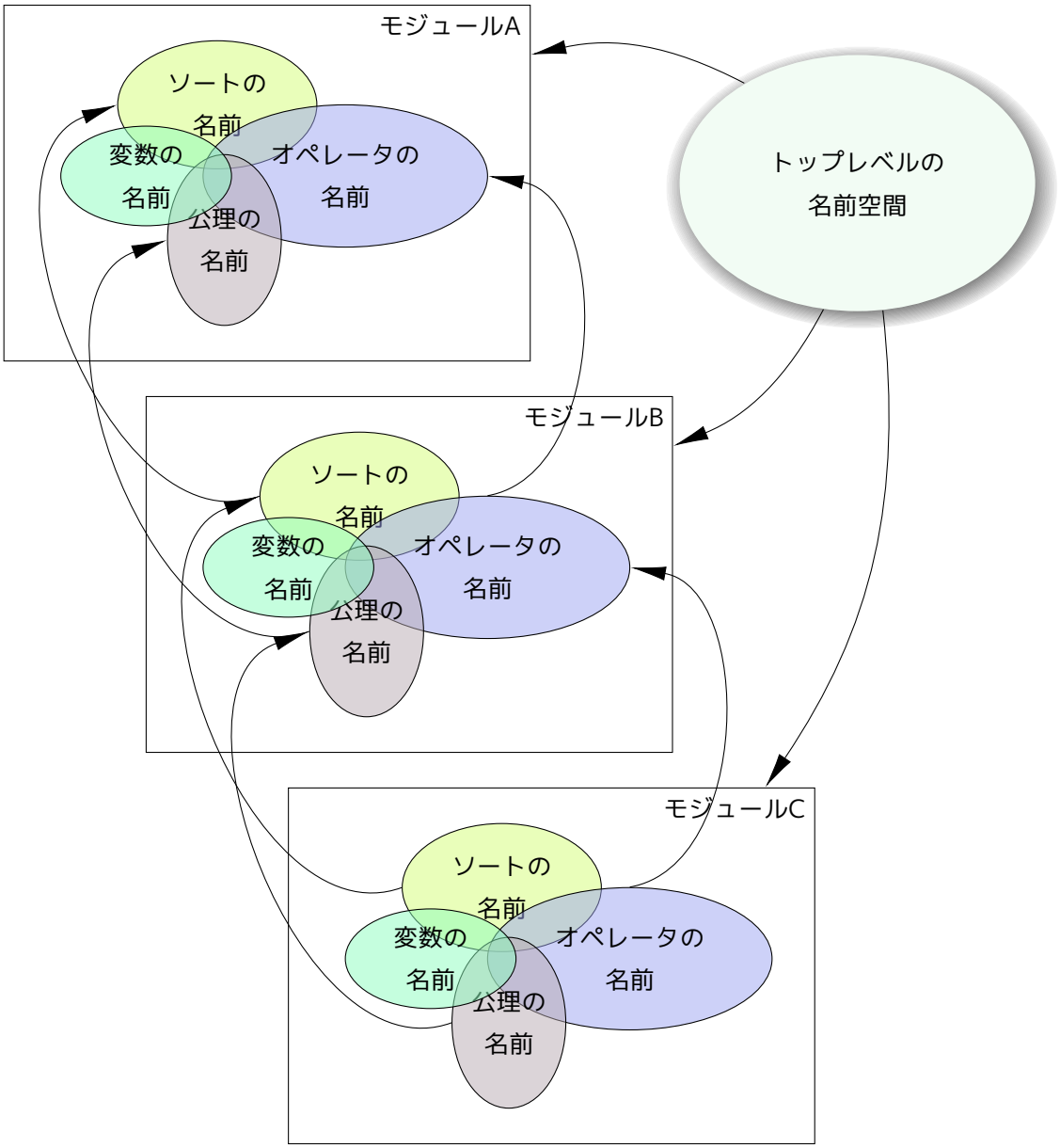

図 1.3: 階層的モジュールの名前空間

輸入されるサブモジュールはモジュール式と呼ばれる構文で指定される． モジュールの名前はモジュール式の構成要素である． モジュール式にはまた，viewの名前が含まれることがある[4]． また，パラメータ付きモジュールのパラメータ宣言部にもモジュール式が出現する． つまり，トップレベルの

[3]一般にモジュールの輸入関係は有向グラフ構造となるが，ここでの議論に関して違いは無い．

[4] モジュール式にはそこに含まれるモジュールの名前で参照されるモジュールの名前空間に含まれるソートやオペレータの名前も含む事が出来るが， モジュール式の中で閉じたものであるため，ここではその詳細に触れない．

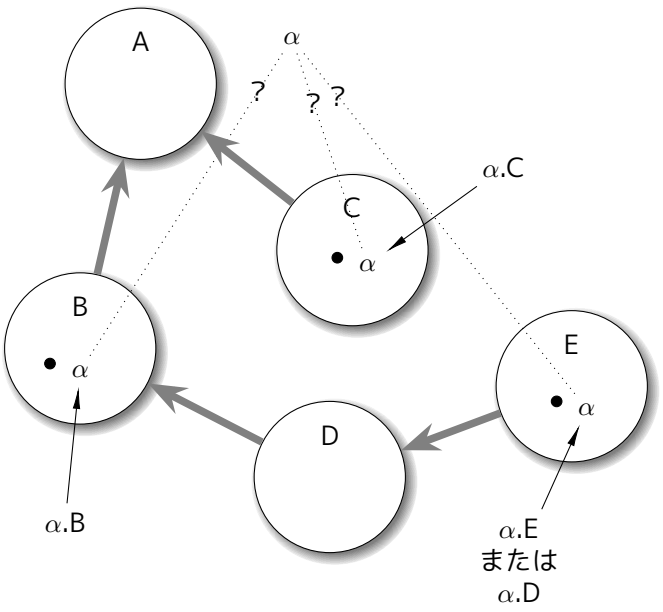

図 1.4: 名前の衝突と文脈限定名による解決

名前空間はモジュール宣言構文の一部(パラメータ宣言部及びモジュール輸入)において可視である．図にはその事も示されている．

パラメータ

パラメータもモジュール輸入の一形態と考えることが出来る[5]． すなわち，パラメータ $C :: M$ があった場合[6]， モジュールMの名前空間にある名前は，やはり(変数を除いて)そのパラメータを持つモジュールの名前空間の一部となる． この場合，あたかもCという名前のモジュールが輸入されていると見なすことが出来る．

文脈限定名による参照の曖昧性の解決

あるモジュールAがサブモジュールをもつ場合， ある名前αの参照が曖昧となる事があり得る． これらが互いに異なったオブジェクトの名前であれば特に問題は無い(変数と定数オペレータの名前の衝突については第 1.1.2を参照)． しかし，例えばαという名前が全てソートの名前だった場合， 構文上同じ文脈に登場することとなるため，どれを指し示すのかが曖昧である(図 1.4を参照)． しかし名前の参照を解決する名前空間を制限すると，曖昧性が解消出来る． 例えば，図 1.4のモジュールEの名前空間にあるαは， モジュールDの名前空間では曖昧性が無い． このような名前空間の制限に対する制限を指定する構文がCafeOBJにはあり，**文脈限定名**と呼ぶ．名前空間の指定にはモジュールの名前を用いる． 例えば名前αについての文脈限定名は $\alpha.M$ の形式で表記され，Mがモジュールの名前で

[5]少なくとも名前空間と名前による参照を議論する場合，そのような解釈で問題ない．
[6]C はパラメータの名前， M はパラメータの仕様を規定するモジュール式．

あり，**モジュール限定子**と呼ぶ． 名前Mで示されるモジュールの名前空間でαを解釈せよ，という意味となる．

文脈限定名$\alpha.M$ のMは，必ずしも参照対象とするオブジェクトの名前αが導入されたモジュールの名前で無くとも良いことに注意されたい． 前述のように，サブモジュールの名前空間は(変数の名前を除き)輸入元の名前空間に 取り込まれるため，名前αの参照について曖昧さが生じないような文脈でありさえすれば良い．

文脈限定名が使える名前

全てのオブジェクトの名前について文脈限定名が使用出来るわけではなく， 使用可能なのは以下3つのオブジェクトのみである:

ソート
定数オペレータ
公理ラベル

表 1.1: 文脈限定名が使用可能なオブジェクト

従ってこれら以外のオブジェクトで同名のものがある場合，曖昧性を解消する手段が無い． そのような場合はモジュール式を用いて名前の付け替えを行い， 曖昧性を回避しなければならない．

1.2 CafeOBJ言語の構文と名前

前章ではCafeOBJの名前空間について，CafeOBJの構文とは切り離して全体的な構造を調べた．本章ではCafeOBJの言語の構文に基づいて，そこに出現する各種の「名前」について洗い出し， 構文上どこにどのような種類の「名前」が存在するかを確認する．

1.2.1 構文表記法

以下でCafeOBJの具体的な構文定義を参照するが，そこで使用される 表中構文は拡張BNF文法で定義されている． 生成規則の形式は次の通り:

$$nonterminal ::= alternative \mid alternative \mid \cdots \mid alternative$$

これは，$nonterminal$ という構文要素は，$\mid$ で区切られた それぞれの $alternative$ のどれかである事を意味する．

非終端シンボルは italic face で，終端記号は “terminal” のような字体で表記する． また終端記号は “ と ” で囲み，拡張BNF文法で使用される メタ文字と区別できるようにする事がある． BNFの拡張形式は以下の通りである:

a ···　1つ以上の a.

a, ···　カンマ記号で区分された一つ以上のa のリスト:
“a” or “a, a” or “a, a, a”, etc.

{ a **}**　**{** と **}** はメタな構文括弧であり，これで囲まれた
a はひとまとまりの構文カテゴリとして扱われる.

[a **]**　a はあっても無くとも良い: “” または “a”.

1.2.2 モジュール宣言の構文と名前

以下にCafeOBJ言語のモジュール宣言の構文のうち「名前」に関係する部分について抜粋したものを示す．構文中 "(1)" のように先頭に番号が振っている構文規則が名前を含んでいるものである．

(1) *module* ::= *module_type module_name* **[** *parameters* **]** **[** *principal_sort* **]**
"{" *module_elt* $\cdots$ "}"
module_type ::= module | module! | module*
(2) *module_name* ::= *ident*
parameters ::= "(" *parameter*, $\cdots$ ")"
(3) *parameter* ::= **[** protecting | extending | including **]** *paramter_name* :: *module_expr*
(4) *parameter_name* ::= *ident*
(5) *principal_sort* ::= principal-sort *sort_name*
module_elt ::= *import* | *sort* | *operator* | *variable* | *axiom* | *macro* | *comment*
import ::= **{** protecting | extending | including | using **}**"(" *module_expr* ")"
sort ::= *visible_sort* | *hidden_sort*
visible_sort ::= "[" *sort_decl*, $\cdots$ "]"
hidden_sort ::= "*[" *sort_decl*, $\cdots$ "]*"
(6) *sort_decl* ::= *sort_name* $\cdots$ **[** *supersorts* $\cdots$ **]**
(7) *supersorts* ::= $<$ *sort_name* $\cdots$
sort_name ::= *sort_symbol***[** *qualifier* **]**
(8) *sort_symbol* ::= *ident*
(9) *qualifier* ::= "."*module_name*
(9) *operator* ::= **{** op | bop **}** *operator_symbol* : **[** *arity* **]** -> *coarity* **[** *op_attrs* **]**
(10) *arity* ::= *sort_name* $\cdots$
(11) *coarity* ::= *sort_name*
op_attrs ::= "{" *op_attr* $\cdots$ "}"
(12) *variable* ::= var *var_name* : *sort_name* | vars *var_name* $\cdots$: *sort_name*
(13) *var_name* ::= *ident*
axiom ::= *equation* | *cequation* | *transition* | *ctransition* | *fol*
equation ::= **{** eq | beq **}** **[** *label* **]** *term* = *term* "."
cequation ::= **{** ceq | bceq **}** **[** *label* **]** *term* = *term* if *term* "."
transition ::= **{** trans | btrans **}** **[** *label* **]** *term* => *term* "."
ctransition ::= **{** ctrans | bctrans **}** **[** *label* **]** *term* => *term* if *term* "."
fol ::= ax**[** *label* **]** *term* "."
(14) *label* ::= "[" *label_name* $\cdots$ "]:"
(15) *label_name* ::= *ident***[** *qualifier* **]**

モジュール宣言文に出現する名前の種別

第 1.2.2節で示した構文はCafeOBJ言語のモジュール宣言文の構文である． ここにはCafeOBJ言語における「名前」のほとんど全て(view の名前を除く，これについては後述する)が出現している．前述のとおり，ここで示した構文定義中で “(N)”のように番号をつけた構文で「名前」が出現している． これらを整理すると以下の通りとなる． これらの名前を使用することにより，利用者は対応するオブジェクトを参照する．

非終端記号	名前種別	参照先	構文番号
$module\_name$	モジュール名	モジュール	(1)
$parameter\_name$	パラメータ名	パラメータモジュール	(3)
$sort\_name$	ソート名	ソート	(5),(6),(7),(10),(11)
$operator\_symbol$	オペレータ名	オペレータ	(9)
var_name	変数名	変数	(12)
$label\_name$	公理ラベル	公理	(15)

上に示した名前種別のうち，operator_symbol(オペレータ名)を除く全ての名前はCafeOBJ 言語の字句定義において ident(識別子) として許される文字列により定義される[7]． 構文(9)のqulifierは第 1.1.2章で説明した文脈限定名を構成するモジュール限定子である． 第 1.1.2章では定数オペレータについても文脈限定名を使用する事が出来ると説明があったが，上記の構文定義中には反映されていない． オペレータ名(operator_symbolについては， 識別子(ident)と合わせて第 1.3章で説明する．

[7]構文番号(2), (4) (8), (13), (14)を参照

1.2.3 モジュール式に出現する名前と名前空間

下はモジュール式(module_exprの構文である.

$$
\begin{array}{ll}
module\_expr & ::= module\_name \mid sum \mid rename \mid instantiation \mid \text{“(”}module\_expr\text{“)”} \\
sum & ::= module\_expr \ \textbf{\{}\ +\ module\_expr\ \textbf{\}}\cdots \\
rename & ::= module\_expr\ \texttt{*}\ \text{“\{”}rename\_map, \cdots \text{“\}”} \\
(1).instantiation & ::= module\_expr\ \text{“(”}\textbf{\{}\ ident[qualifier]\ \texttt{<=}\ aview\textbf{\}},\ \cdots\ \text{“)”} \\
rename\_map & ::= sort\_map \mid op\_map \\
(2).sort\_map & ::= \textbf{\{}\ \texttt{sort} \mid \texttt{hsort}\ \textbf{\}}\ sort\_name\ \texttt{->}\ ident \\
(3).op\_map & ::= \textbf{\{}\ \texttt{op} \mid \texttt{bop}\ \textbf{\}}\ op\_name\ \texttt{->}\ operaotr\_symbol \\
(4).op\_name & ::= operator\_symbol \mid \text{“(”}operator\_symbol\text{“)”}qualifier \\
(5).aview & ::= view\_name \mid module\_expr \\
 & \quad \mid\ \texttt{view to}\ module\_expr\ \text{“\{”}view\_elt, \cdots \text{“\}”} \\
(6).view\_name & ::= ident \\
view\_elt & ::= sort\_map \mid op\_view \mid variable \\
op\_view & ::= op\_map \mid term\ \texttt{->}\ term
\end{array}
$$

1.2.4 view 宣言に出現する名前と名前空間

$$
\begin{array}{l}
view ::= \texttt{view}\ view\_name\ \texttt{from}\ module\_expr\ \texttt{to}\ module\_expr \\
\qquad \text{“\{”}\ view\_elt, \cdots\ \text{“\}”}
\end{array}
$$

1.3 字句区切り文字と識別子，オペレータ名

識別子とオペレータ名はともに，不可視な文字を含まない連続したASCII英数字の文字列(**字句**) から構成される． CafeOBJ システムは入力文字列を読み込む際に字句の区切りとして空白文字，タブ，改行，改ページ文字を 区切り記号として認識する．このような文字を**字句区切り文字**と呼ぶ． すなわち，入力文字列は字句区切り文字によって切断される． 空白文字，タブ，改行，改ページのような不可視な文字は入力の際にスキップされ， 単なる字句の区切り記号として扱われる．これらに加えて以下の7文字が 字句区切り文字として定義されており，これらの文字はそれ自身で一つの字句を構成するものとして 認識される:

```
1. (          -- left paren
2. )          -- right paren
3. [          -- left square bracket
4. ]          -- right square bracket
5. {          -- left brace
6. }          -- right brace
7. ,          -- comma
```

1.3.1 識別子

字句区切り文字は入力文字列から字句を切り出す印となる文字であるという性質上，ソート名やモジュール名等の識別子を構成する文字として用いることは出来ない[8].

1.3.2 オペレータ名

一方，オペレータ名を構成する文字要素は上で述べた7つの可視な字句区切り文字のうち，1の丸括弧を除いてオペレータ名の一部として用いることが可能である[9]. これらに加えて，空白文字もオペレータ名の一部として使用可能である. 例えば下は，”this is an operator” という名前の2引数のオペレータの宣言である.

```
module FOO {
  [ S ]
  op this is an operator : S S -> S
}
-- defining module FOO..* done.
CafeOBJ> select FOO
FOO> sh op this       is an        operator
.......................(this is an operator).......................
  * rank: S S -> S
  - attributes:  { prec: 0 }
FOO>
```

上の例を見て了解されるように，コマンドでオペレータ名を表記する際はオペレータ名を構成する連続した空白文字は無視され，1文字として扱われる.

また，mixfix オペレータの宣言において，項を表記する際の引き数位置を示すための下線(_)はオペレータ名の一分として扱われる. すなわち，コマンド類でオペレータを参照する際は下線(_)も含めて指定する必要がある. コマンドの引数としてオペレータ名を指定する場合は次に示す例のように，下線(_)とそれ以外の文字との間に空白文字をおいても良い.

[8] システムはこれらの文字を単純に自己自身で字句を構成する文字として読み込む. そのため，文脈によってはエラーとならず，識別子を構成する文字として受け入れているように
見える場合がある. 例えば“[{SortName}]”というソート宣言はエラーとならない. しかし，実際にはこれは3つのソート “{”, “Sort”, “}” が宣言されたものとして認識される.

[9] 丸括弧はどの文脈においても“グルーピング”を表現するメタ文字として取り扱われる.

```
CafeOBJ> select NAT
NAT> sh op _+_
...............(_ + _) default attributes { assoc comm }...............
  * rank: NzNat NzNat -> NzNat
  - attributes:  { assoc comm demod prec: 33 r-assoc }
    - axioms:
      eq [:BDEMOD]: (NN:NzNat + NM:NzNat)
       = #! (+ nn nm)
      eq [:BDEMOD]: (M + N) = #! (+ m n)
  * rank: Nat Nat -> Nat
  - attributes:  { assoc comm idr: 0 demod prec: 33 r-assoc }
    - axioms:
      eq [:BDEMOD]: (AC:Nat + (NN:NzNat + NM:NzNat))
       = (AC + #! (+ nn nm))
      eq [:BDEMOD]: (M + N) = #! (+ m n)
      eq [ident1]: (0 + X-ID:Nat) = X-ID
NAT> sh op _ + _
...............(_ + _) default attributes { assoc comm }...............
  * rank: NzNat NzNat -> NzNat
  - attributes:  { assoc comm demod prec: 33 r-assoc }
    - axioms:
      eq [:BDEMOD]: (NN:NzNat + NM:NzNat)
       = #! (+ nn nm)
      eq [:BDEMOD]: (M + N) = #! (+ m n)
  * rank: Nat Nat -> Nat
  - attributes:  { assoc comm idr: 0 demod prec: 33 r-assoc }
    - axioms:
      eq [:BDEMOD]: (AC:Nat + (NN:NzNat + NM:NzNat))
       = (AC + #! (+ nn nm))
      eq [:BDEMOD]: (M + N) = #! (+ m n)
      eq [ident1]: (0 + X-ID:Nat) = X-ID
NAT>
```

1.4　字句区切り文字の設定

　システムに組み込まれた字句区切り文字を追加するためのコマンドを新たに導入した． 本コマンドは新規の字句区切り文字の導入によるインパクトを評価するために用意したものであり， 将来的なCafeOBJ言語の字句区切り文字の拡張のための試験用に利用することを想定している．

```
<Delimiterコマンド> ::= delimiter { + | - } "{" <文字> ... "}"
```

+が指定された場合，<文字>で指定した文字を新たな字句など区切り文字として指定する． 逆に，新

たに導入された字句区切り文字をキャンセルするには-を指定する． 本コマンドによって，システムに組み込みの字句区切り文字は影響を受けない(組み込みの字句区切り文字を -指定によって字句区切り文字で無くすることは出来ない．

1.5 モジュールの名前空間の表示

1.5.1 namesコマンド

今回新たに導入された **names** コマンドによって， 指定のモジュール内で可視の名前の一覧を見ることが出来る． 構文は下の通り:

```
<名前表示コマンド>  ::= names [<モジュール式>] .
```

<モジュール式>を省略すると，現在の文脈(モジュール)で可視の名前を表示する． 名前の表示はアルファベット順が原則であるが， mixfix オペレータについては最後にまとめて表示される．

例: 下の例のモジュールFOOは組み込みモジュールのNATを輸入している． その際に，ソートNatをNaturalに，ソート_+_ をplusと名前を付け替えている． また，モジュール輸入構文で新たに導入された**as**指定によって**NAT *{sort ....}**のモジュールをNATURALという名前で参照できるようにしている．

```
CafeOBJ> mod FOO { inc as NATURAL (NAT *{sort Nat -> Natural, op _+_ -> plus}) }
-- defining module FOO
; Loading /usr/local/cafeobj-1.4/lib/nat.bin
-- defining module! NAT
....
```

上のモジュールFOOに対して**names**コマンドによって可視の名前を表示した結果を以下に示す．

```
CafeOBJ> names FOO .

** [*] ----------------------------------------
*Cosmos*
  - sort declared in CHAOS:UNIVERSAL
*HUniversal*
  - hidden sort declared in CHAOS:UNIVERSAL
*Universal*
  - sort declared in CHAOS:UNIVERSAL
** [:] ----------------------------------------
:BDEMOD
  - axiom declared in NZNAT * { ... }
    eq [:BDEMOD]: (s NN:NzNat) = #! (1+ nn)
  - axiom declared in NZNAT * { ... }
    eq [:BDEMOD]: (NN:NzNat >= NM:NzNat)
     = #! (>= nn nm)
  - axiom declared in NZNAT * { ... }
    eq [:BDEMOD]: (NN:NzNat > NM:NzNat)
     = #! (> nn nm)
  - axiom declared in NZNAT * { ... }
    eq [:BDEMOD]: (NN:NzNat <= NM:NzNat)
     = #! (<= nn nm)
  - axiom declared in NZNAT * { ... }
    eq [:BDEMOD]: (NN:NzNat < NM:NzNat)
     = #! (< nn nm)
  - axiom declared in NZNAT * { ... }
    eq [:BDEMOD]: (NN:NzNat quot NM:NzNat)
     = #! (if (> nn nm) (truncate nn nm) 1)
  - axiom declared in NZNAT * { ... }
    eq [:BDEMOD]: (NN:NzNat * NM:NzNat)
     = #! (* nn nm)
  - axiom declared in NZNAT * { ... }
    eq [:BDEMOD]: d(NN:NzNat,NM:NzNat)
     = #! (if (= nn nm) 1 (abs (- nn nm)))
  - axiom declared in NZNAT * { ... }
    eq [:BDEMOD]: plus(NN:NzNat,NM:NzNat)
     = #! (+ nn nm)
  - axiom declared in NATURAL
    eq [:BDEMOD]: (p NN:NzNat) = #! (- nn 1)
  - axiom declared in NATURAL
    eq [:BDEMOD]: (s 0) = 1
  - axiom declared in NATURAL
    eq [:BDEMOD]: (N:Natural >= 0) = true
  - axiom declared in NATURAL
```

```
  eq [:BDEMOD]: (0 >= NN:NzNat) = false
- axiom declared in NATURAL
  eq [:BDEMOD]: (NN:NzNat > 0) = true
- axiom declared in NATURAL
  eq [:BDEMOD]: (0 > N:Natural) = false
- axiom declared in NATURAL
  eq [:BDEMOD]: (0 <= N:Natural) = true
- axiom declared in NATURAL
  eq [:BDEMOD]: (NN:NzNat <= 0) = false
- axiom declared in NATURAL
  eq [:BDEMOD]: (0 < NN:NzNat) = true
- axiom declared in NATURAL
  eq [:BDEMOD]: (N:Natural < 0) = false
- axiom declared in NATURAL
  eq [:BDEMOD]: (NN:NzNat divides M:Natural)
   = #! (= 0 (rem m nn))
- axiom declared in NATURAL
  eq [:BDEMOD]: (M:Natural rem NN:NzNat)
   = #! (rem m nn)
- axiom declared in NATURAL
  eq [:BDEMOD]: (M:Natural quo NN:NzNat)
   = #! (truncate m nn)
- axiom declared in NATURAL
  eq [:BDEMOD]: (N:Natural * 0) = 0
- axiom declared in NATURAL
  eq [:BDEMOD]: plus(M:Natural,N:Natural)
   = #! (+ m n)
- axiom declared in NATURAL
  eq [:BDEMOD]: sd(M:Natural,N:Natural)
   = #! (abs (- m n))
```

```
** [B] ----------------------------------------
BASE-BOOL
  - indirect sub-module
BOOL
  - indirect sub-module
Bool
  - sort declared in TRUTH-VALUE
  - operator:
    op Bool : -> SortId { constr prec: 0 }
        -- declared in module TRUTH-VALUE
** [C] ----------------------------------------
CHAOS:PARSER
  - indirect sub-module
CHAOS:UNIVERSAL
  - indirect sub-module
** [D] ----------------------------------------
d
  - operator:
    op d : NzNat NzNat -> NzNat { comm demod prec: 0 }
        -- declared in module NZNAT * { ... }
** [E] ----------------------------------------
EQL
  - indirect sub-module
** [F] ----------------------------------------
false
  - operator:
    op false : -> Bool { constr prec: 0 }
        -- declared in module TRUTH-VALUE
** [I] ----------------------------------------
ident0
  - axiom declared in NZNAT * { ... }
    eq [ident0]: (1 * X-ID:NzNat) = X-ID
ident1
  - axiom declared in NATURAL
    eq [ident1]: plus(0,X-ID:Natural)
     = X-ID
ident2
  - axiom declared in NATURAL
    eq [ident2]: (1 * X-ID:Natural)
     = X-ID
```

```
** [N] ----------------------------------------
Natural
  - sort declared in NAT-VALUE * { ... }
  - operator:
    op Natural : -> SortId { constr prec: 0 }
        -- declared in module NAT-VALUE * { ... }
NATURAL
  - direct sub-module, alias of module NAT *{  ...  }
NzNat
  - sort declared in NZNAT-VALUE
  - operator:
    op NzNat : -> SortId { constr prec: 0 }
        -- declared in module NZNAT-VALUE
NZNAT-VALUE
  - indirect sub-module
** [P] ----------------------------------------
plus
  - operator:
    op plus : Natural Natural -> Natural { assoc comm idr: 0 demod
                                           prec: 0 r-assoc }
        -- declared in module NATURAL
    op plus : NzNat NzNat -> NzNat { assoc comm demod prec: 0 r-assoc }
        -- declared in module NZNAT * { ... }
** [S] ----------------------------------------
sd
  - operator:
    op sd : Natural Natural -> Natural { comm demod prec: 0 }
        -- declared in module NATURAL
SortId
  - sort declared in CHAOS:PARSER
  - operator:
    op SortId : -> SortId { constr prec: 0 }
        -- declared in module CHAOS:PARSER
SyntaxErr
  - sort declared in CHAOS:PARSER
  - operator:
    op SyntaxErr : -> SortId { constr prec: 0 }
        -- declared in module CHAOS:PARSER
```

```
** [T] ----------------------------------------
true
  - operator:
    op true : -> Bool { constr prec: 0 }
        -- declared in module TRUTH-VALUE
TRUTH
  - indirect sub-module
TRUTH-VALUE
  - indirect sub-module
TypeErr
  - sort declared in CHAOS:PARSER
  - operator:
    op TypeErr : -> SortId { constr prec: 0 }
        -- declared in module CHAOS:PARSER
** [Z] ----------------------------------------
Zero
  - sort declared in NAT-VALUE * { ... }
  - operator:
    op Zero : -> SortId { constr prec: 0 }
        -- declared in module NAT-VALUE * { ... }
** [_] ----------------------------------------
_ Bottom _
  - sort declared in CHAOS:UNIVERSAL
_ HBottom _
  - hidden sort declared in CHAOS:UNIVERSAL
(if _ then _ else _ fi)
  - operator:
    op if _ then _ else _ fi : Bool *Cosmos* *Cosmos* -> *Cosmos* { strat: (1 0)
                                                                    prec: 0
                                                                    }
        -- declared in module TRUTH
(not _)
  - operator:
    op not _ : Bool -> Bool { strat: (0 1) prec: 53 }
        -- declared in module BASE-BOOL
(p _)
  - operator:
    op p _ : NzNat -> Natural { demod prec: 15 }
        -- declared in module NATURAL
(parsed:[ _ ], rest:[ _ ])
  - operator:
    op parsed:[ _ ], rest:[ _ ] : *Universal* *Universal* -> SyntaxErr
                                  { prec: 0 }
        -- declared in module CHAOS:PARSER
```

```
(s _)
  - operator:
    op s _ : NzNat -> NzNat { demod prec: 15 }
        -- declared in module NZNAT * { ... }
    op s _ : Natural -> NzNat { demod prec: 15 }
        -- declared in module NATURAL
(_ * _)
  - operator:
    op _ * _ : NzNat NzNat -> NzNat { assoc comm idr: 1 demod prec: 31
                                      r-assoc }
        -- declared in module NZNAT * { ... }
    op _ * _ : Natural Natural -> Natural { assoc comm idr: 1 demod
                                            prec: 31 r-assoc }
        -- declared in module NATURAL
(_ := _)
  - operator:
    pred _ := _ : *Cosmos* *Cosmos*  { prec: 51 }
        -- declared in module TRUTH
(_ :is _)
  - operator:
    pred _ :is _ : *Cosmos* SortId  { prec: 125 }
        -- declared in module TRUTH
(_ < _)
  - operator:
    pred _ < _ : NzNat NzNat  { demod prec: 51 }
        -- declared in module NZNAT * { ... }
    pred _ < _ : Natural Natural  { demod prec: 51 }
        -- declared in module NATURAL
(_ <= _)
  - operator:
    pred _ <= _ : NzNat NzNat  { demod prec: 51 }
        -- declared in module NZNAT * { ... }
    pred _ <= _ : Natural Natural  { demod prec: 51 }
        -- declared in module NATURAL
(_ = _)
  - operator:
    pred _ = _ : *Cosmos* *Cosmos*  { comm prec: 51 }
        -- declared in module EQL
```

```
(_ =*= _)
  - operator:
    pred _ =*= _ : *HUniversal* *HUniversal*  { prec: 51 }
        -- declared in module TRUTH
(_ =/= _)
  - operator:
    pred _ =/= _ : *Cosmos* *Cosmos*  { prec: 51 }
        -- declared in module TRUTH
(_ == _)
  - operator:
    pred _ == _ : *Cosmos* *Cosmos*  { prec: 51 }
        -- declared in module TRUTH
(_ =b= _)
  - operator:
    pred _ =b= _ : *Cosmos* *Cosmos*  { prec: 51 }
        -- declared in module TRUTH
(_ > _)
  - operator:
    pred _ > _ : NzNat NzNat  { demod prec: 51 }
        -- declared in module NZNAT * { ... }
    pred _ > _ : Natural Natural  { demod prec: 51 }
        -- declared in module NATURAL
(_ >= _)
  - operator:
    pred _ >= _ : NzNat NzNat  { demod prec: 51 }
        -- declared in module NZNAT * { ... }
    pred _ >= _ : Natural Natural  { demod prec: 51 }
        -- declared in module NATURAL
(_ and _)
  - operator:
    op _ and _ : Bool Bool -> Bool { assoc comm prec: 55 r-assoc }
        -- declared in module BASE-BOOL
(_ and-also _)
  - operator:
    op _ and-also _ : Bool Bool -> Bool { strat: (1 0 2) prec: 55
                                          r-assoc }
        -- declared in module BASE-BOOL
```

```
(_ divides _)
  - operator:
    pred _ divides _ : NzNat Natural  { demod prec: 51 }
        -- declared in module NATURAL
(_ iff _)
  - operator:
    op _ iff _ : Bool Bool -> Bool { strat: (0 1 2) prec: 63 r-assoc }
        -- declared in module BASE-BOOL
(_ implies _)
  - operator:
    op _ implies _ : Bool Bool -> Bool { strat: (0 1 2) prec: 61
                                         r-assoc }
        -- declared in module BASE-BOOL
(_ or _)
  - operator:
    op _ or _ : Bool Bool -> Bool { assoc comm prec: 59 r-assoc }
        -- declared in module BASE-BOOL
(_ or-else _)
  - operator:
    op _ or-else _ : Bool Bool -> Bool { strat: (1 0 2) prec: 59
                                         r-assoc }
        -- declared in module BASE-BOOL
(_ quo _)
  - operator:
    op _ quo _ : Natural NzNat -> Natural { demod prec: 41 }
        -- declared in module NATURAL
(_ quot _)
  - operator:
    op _ quot _ : NzNat NzNat -> NzNat { demod prec: 31 l-assoc }
        -- declared in module NZNAT * { ... }
(_ rem _)
  - operator:
    op _ rem _ : Natural NzNat -> Natural { demod prec: 31 l-assoc }
        -- declared in module NATURAL
(_ xor _)
  - operator:
    op _ xor _ : Bool Bool -> Bool { assoc comm prec: 57 r-assoc }
        -- declared in module BASE-BOOL
CafeOBJ>
```

1.5.2 look up コマンド

look up コマンドは，指定のあるいは現在文脈のモジュールにおいて，ある名前で参照されるものが何であるかを表示する． 構文は次の通りである．

```
<LOOK UPコマンド>    ::= lookup [in <モジュール名> :] <識別子> | <オペレータ名> .
```

例:この例は，先に第 1.5.1章で見たモジュールFOO内で**plus**という名前で参照されるものが何であるかを表示したものである．

```
CafeOBJ> look up in FOO : plus .

** [P] ----------------------------------------
plus
  - operator:
    op plus : Natural Natural -> Natural { assoc comm idr: 0 demod
                                           prec: 0 r-assoc }
        -- declared in module NATURAL
    op plus : NzNat NzNat -> NzNat { assoc comm demod prec: 0 r-assoc }
        -- declared in module NZNAT * { ... }
CafeOBJ> select FOO
FOO> look up Natural
.

** [N] ----------------------------------------
Natural
  - sort declared in NAT-VALUE * { ... }
  - operator:
    op Natural : -> SortId { constr prec: 0 }
        -- declared in module NAT-VALUE * { ... }
FOO>
```