K 言語文法マニュアル
(Kuroyuri 2005 年 11 月中旬バージョン向け)

JAIST StarBED **チーム**

2005 年 11 月中旬

目次

第1章　導　入

この文章は K 言語の文法マニュアルである。ネットワーク実験駆動システム Kuroyuri で用いられることを念頭に設計されている。この文章では文法にのみ焦点を当てている。Kuroyuri やそれに関するプログラムの集合体である SpringOS に関しては、別の文献を参照して欲しい。

1.1 連 絡 先

この文章に関する質問は以下の電子メールアドレスに電子メールを送るとよい。奇妙な動作を見付けた場合にも、状況を伝えて欲しい。なお、使ったプログラムに関する説明 (tar.gz ファイル名等) をつけるとアクションが早くなるだろう。

`info@starbed.org`

1.2 用 語

実験: コンピュータ機材を試験的に操作すること。この文章ではネットワークを中心に扱う実験を想定している。

ユーザ: 実験者 (複数名も可)。

プロジェクト: 実験者がいくつも実験をする際に、個々の実験をプロジェクトと呼ぶ。

ノード: 実験に登場する実体。この文章では PC と考えて良い。

評価器: K 言語を扱う言語処理系。この文章では kuroyuri に含まれる master と slave を対象に想定している。特に K 言語を読み込むのは master である。

1.3 管 理 カ テ ゴ リ

一般的に実験施設上では多くの実験が同時に進行する。その際には実験間での資源の衝突等に気を付けねばならない。一人で占有している (複数ユーザで共有しない) 場合は、さほど気にしなくて良いだろう。

1.4　記 述 の 基 本 的 規 則

K 言語は行指向の言語である。

行頭が # (シャープ; shape) の行はコメントとして扱われる。以下の例では最後の行のみ有効となる。

```
# this is comment, please ignore.
print "helo"
```

数字やシンボルはそのまま記述することができる。文字列は ” (ダブル・クォート; double quote) で囲む必要がある。

```
1323
abc
"broadway"
```

行が長くなった場合は \ (バック・スラッシュ; back-slash) で次の行と連結することを指示する。

```
jugemu jugemu goko no surikire kaijari suigyo no suigyoumatsu \
unraimatsu furaimatsu kuneru tokoro ni sumu tokoro yaburakouji \
no burakouji
```

変数は = (イコール; equal) で新たに作ることができる。

```
a = 123
b = "www server"
```

詳細は付録 B(39 ページ) に譲るが、いくつか予約語があるので関数や変数の名前として使わないよう注意が必要である。特に、初心者が変数に t を使う場合が多いようなので注意してほしい。

第2章　状 況 お よ び 環 境

この章では実験の状況を記述する構文を紹介する。

2.1　実験実施者とプロジェクト名

`user` <”*name*”> <”*mail-address*”>	`user`
担当者の名前および連絡先を宣言する。	

`project` <”*name*”>	`project`
プロジェクト名を宣言する。	

`encd ipaddr` <”*address*”> `port` <”*number*”>	`encd`
実験全体のノード設定駆動プログラム (experiment node configuration driver;ENCD) に関する設定。これは kuroyuri のマスターを指す。したがって、kuroyuri のマスターを実行するノードの IP アドレスを指定する。一般的には管理ネットワークの IP アドレスを用いる。ノードから複数のネットワークで到達できる場合は、それ以外のネットワークを用いることも可能。	

2.2　ネ ッ ト ワ ー ク

`ipaddrrange` <”*ipaddr*[*/mask-length*]”>	`ipaddrrange`
この実験で利用する IP アドレスの範囲を指定する。実験記述中にこの範囲に含まれない IP アドレスが登場した場合に警告が出力される。	

2.3　グローバル・シナリオ

`scenario` <*block*>	`scenario`
シナリオの記述。5 参照。	

2.4　施 設 関 連 サ ー バ

以下、ipaddr は IP アドレス、ipaddrrange は IP アドレス範囲、port は TCP ポート番号を示す。

`rmanager ipaddr` <”*address*”> `port` <”*number*”>	`rmanager`
リソースマネージャ(resource manager; RM) をアドレスとポート番号で指定する。ボート番号が文字列であることに注意。	

`wolagent ipaddr` <”*address*”> `port` <”*number*”> `ipaddrrange` <”*address/len*”>	`wolagent`
WoL(Wake On LAN) エージェントとその担当アドレス範囲を指定する。この構文は実験記述中に複数個記述できる。	

```
wolagent ipaddr "172.16.1.101" port "5959" ipaddrrange "172.16.1.0/23"
wolagent ipaddr "172.16.3.101" port "5959" ipaddrrange "172.16.3.0/23"
```

fncp	fncp ipaddr <”*address*”> port <”*number*”>
	施設全体のノード設定案内プログラム (facility node configuration pilot;FNCP) に関する設定。
tftpdman	tftpdman ipaddr <”*address*”> port <”*number*”>
	Kuroyuri は TFTP を用いてディスクイメージをインストールしている。TFTPD のディレクトリを担当している、ディレクトリ制御プログラム (direcotry manipulator; DMAN) に関する設定。
tftpdir	tftpdir <”*path*”>
	ディスクイメージを格納しているディレクトリを指示する。
nidiskimage	nidiskimage <”*path*”>
nikernel	nikernel <”*path*”>
nipxeloader	nipxeloader <”*path*”>
	ディスクイメージをインストールするプログラム NI を起動するために、NI に用いる OS のディスクイメージ、カーネル、PXE ローダを指定する[1] 。

[1]従来の pxeloader を変更

`swtype name <”name”> type <”type”>` `swtype`

スイッチ名と種類を指定する。現在サポートしているのは IOS,CATOS そして IronWare である。

あとえば、ある施設では以下のような設定が記述される。

```
swtype name "silaswa001" type "IOS"
swtype name "silaswb001" type "CATOS"
swtype name "silaswb002" type "CATOS"
swtype name "silaswb003" type "IronWare"
```

第3章　ノ　ー　ド

ノードとはネットワーク実験上のエンティティを指す。ここでは、PC と考えて良い。

```
node c {
    method "HDD"
    disktype "IDE"
    partition 2
    ostype "FreeBSD"
    diskimage "ftp://172.16.210.9/ifcheck-e.gz"
    netif media fastethernet
    scenario {
        recv x
        msgswitch x {
        "start" {
            wakewait "/sim/netperf" "/sim/netperf" "-H" "192.168.3.1"
            send "finishperf"
          }
        }
    }
}
```

---

node <*name*> <*block*>　　node

<block> には以下のような属性を記述する。

3.1 ノ　ー　ド　属　性

---

method "HDD"|"direct"　　method

ノードの操作方法を指定する[1]。HDD はハードディスクによる起動を意味し、以下の disktype や partition 属性に従って、ノードを初期化して利用する。

direct は直接通信を意味し、agent 属性に従って、直接スレーブと通信する。この場合は何らかの方法で対象ノードのスレーブを起動せねばならない。

---

disktype "IDE"|"SCSI"　　disktype

ディスクのタイプ。現状では IDE と SCSI を指定できる。

---

partition <*number*>　　partition

利用するパーティション番号。利用できるパーティションは施設の管理者に問い合わせると良いだろう。

[1]従来の Memory は使用停止

ちなみに、StarBED のシミュレーションクライアント装置では以下のようになっている。

番号	摘要
1	WINDOWS 2000
2	FreeBSD
3	なし (FAT32 でフォーマットしてある)
4	拡張パーティション
5	TurboLinux

`ostype`

`ostype` <*"ostype"*>

利用する OS の種類。現状では機能しない。

`diskimage`

`diskimage` <*"URL"*>

利用するディスクイメージを URL で指定する。NI が認識できる形式で指定する必要がある。現在のところ、NI が対応しているのは FTP と HTTP である。

`netif`

`netif media fastethernet`|`gigabitethernet`

ネットワークインターフェースを宣言。メディアはファスト・イーサネットとギガビット・イーサネットに対応している。

`agent`

`agent ipaddr` <*ipaddr*> `port` <*port*>

特定のノード上の評価器をノードとして扱うよう指示する。通常はこの評価器はスレーブを意味する。多くの場合はリソースマネージャを通して施設中の PC が自動的にノードに割当てられるため、この構文を用いることはないだろう。

`scenario`

`scenario` <*block*>

ノード・シナリオの記述。後述 (第 5 章)。

3.2 ノード・クラス

実験では属性の同じノードを多数利用する場合が多いので、同じ属性のノード集合を作る方法を用意してある。

`nodeclass` <*name*> <*block*> `nodeclass`

クラスの構文は node と同様であるが、実験記述では直接操作する実体ではない。

`nodeset` <*name*> `class` <*class-name*> `num` <*number*> `nodeset`

`nodeset` <*name*> `class` <*class-name*> `num` <*number*> `spare` <*number*>

---

class-name で宣言したノード・クラスと同じ属性のノード集合を宣言する。このノード集合は num で指定した個数のノードで構成される。予備機は spare で指定した個数となる。これは sparenodemin, sparenoderatio より高い優先度で適用される。指定が無い場合は sparenodemin, sparenoderatio に従う。

ノード群を構成するノードの属性の変更は以下のように行う。

```
nodeset frontend class serverC num 5
...
frontend[4].netif[4].ipaddr = "192.168.9.41"
```

3.3 セットアップ・パラメータ

`sparenodemin` <*integer*> `sparenodemin`

予備ノードの最小値を指定する。標準値は 1 である。

`sparenoderatio` <*integer*> `sparenoderatio`

確保するノードの比率を指定する。標準値は 105 である。

最終的には、n 台のノードが必要な際には、MIN(n+sarenodemin,n*sparenoderatio/100) の結果が確保するノードとなる。

したがって予備機の確保を抑制する場合は以下のような記述が必要となる。

```
sparenodemin 0
sparenoderatio 100
```

`setuptimeout total` <*second*> `setuptimeout`

`setuptimeout total` <*second*> `warm` <*second*>

ノードの起動処理を打ち切る時間を指定する。単位は秒。total はノード起動処理全体を打ち切る。warm は必要最小限のノードが揃った後に待つ時間。

```
setuptimeout total 1800 warm 6
```

第4章　ネットワーク

ネットワークはノードと似た形態で宣言する。ノード同様にクラス定義とクラスを用いた集合宣言も可能である。

```
netclass e {
    media fastethernet
    ipaddrrange "192.168.3.0/24"
}
```

net <*name*> <*block*>	net
netclass <*name*> <*block*>	netclass
netset <*name*> class <*classname*> num <*num*>	netset

ネットワークの属性は以下で説明する。

4.1 ネットワーク属性

media fastethernet|gigabitethernet	media

ネットワークのメディア。

ipaddrrange <"*ipaddr/*[*mask-length*]">	ipaddrrange

このネットワークに割当てる IP アドレスの範囲。

4.2 接続と開放

ネットワークへのノードの論理的な接続と開放を記述する構文が用意されている。

attach <*netif*> <*net*>	attach

このネットワークにノードのインターフェースの接続を示す。この構文で、評価器はそのノードの参加するネットワークを認識する。IP アドレスは参加したネットワークの属性から自動的に割り当てられる。そして、ネットワークスイッチの設定も自動的に施される。

```
attach cl.netif[2] leaf
```

detach <*netif*> <*net*>	detach

detach は attach と反対にインターフェースをネットワークから切り放す。

```
detach cl.netif[2] leaf
```

第5章　シ　ナ　リ　オ

シナリオはノードの挙動の集まりで、各ノードの属性として定義すると、そのノードの挙動 (ノードシナリオ; node scenario) となり、それ以外は、実験全体の挙動 (グローバルシナリオ; global scenario) となる。

双方とも記述された順に実行される。

`scenario` <*block*>	`scenario`

ブロックでは以下の文が利用できる。

5.1　外部プログラム起動

シナリオの大部分は実験対象となるソフトウェアの設定と起動に費される。したがって、対象プログラムや設定プログラムの起動は重要な構文である。

`wake` <*"program-path"*> <*"program"*> <*"args"*>...	`wake`
`wakewait` <*"program-path"*> <*"program"*> <*"args"*>...	`wakewait`

wake と wakewait はプログラムを起動する。wakewait は起動したプログラムの終了を待つ点が wait と異なる。program-path は起動対象のプログラムの絶対パスである。program はプログラムに与えるプログラムの名称 (名称で挙動の変わるプログラムがあるため必要) である。args は引数となる。

コンテキストの面で見ると、wake は並行実行、wakewait は直列実行となる。

wakewait を用いる場合は終了条件を考える必要がある。たとえば、多くの OS の ping は特別な引数を与えない場合には自動的には停止しない。そのような ping には -c を試みると良いだろう。

```
wakewait "/usr/bin/ping" "/usr/bin/ping" "-c" "10" "www.starbed.org"
```

自動停止しないプロセスを wake で起動した場合は、kill や killall を使って停止することになる。

```
wake "/usr/bin/ping" "/usr/bin/ping" "anywhere.onthe.net"
sleep 10
wakewait "/bin/killall" "/bin/killall" "ping"
```

簡単に停止しない場合は、数秒後に強制的に停止させることも必要だろう。

```
wakewait "/bin/killall" "/bin/killall" "foo"
sleep 3
wakewait "/bin/killall" "/bin/killall" "-KILL" "foo"
```

名称で挙動の変わるプログラムは少ないため、wake と wakewait を簡略化した構文も用意した。それが、call と callw である。これらはプログラム名を指定しない構文で、program-path をそのままプログラム名にした wake, wakewait と等価である。

call	call <*"program-path"*> <*"args"*>...
callw	callw <*"program-path"*> <*"args"*>...

5.1.1 リダイレクト

wake や wakewait で起動したプログラムの出力をリダイレクトする文法も用意してある。

記号	意味
>	標準出力 (stdout) をリダイレクト
>>	標準出力をリダイレクト (追記)
2>&1	標準エラー (stderr) を標準出力へ混ぜる

たとえば、以下のような文で ps と ls の実行結果を集めることができる。

```
wakewait "/bin/ps" "/bin/ps" "aux" > "/tmp/slave.out"
wakewait "/bin/ls" "/bin/ls" "-l" "/" >> "/tmp/slave.out"
```

5.2 プロセス停止

kill	kill <*pid*>

指定したプロセスに指定したシグナルを発行する。シグナルは伝統にしたがって SIGTERM を用いる。

```
cid = wake "/usr/bin/ping" "/usr/bin/ping" "anywhere.onthe.net"
sleep 10
kill cid
```

killval	killval <*signal*> <*pid*>

指定したプロセスに指定したシグナルを発行する。signal が数字の場合は、その数字をシグナル番号としてシグナルを発行する。signal が文字列の場合は、シグナル番号に変換する。シグナル番号は OS によって異なるので、文字列を使った方が汎用性が高い。

```
cid = wake "/usr/bin/ping" "/usr/bin/ping" "anywhere.onthe.net"
sleep 10
killval "TERM" cid
```

番号を指定する場合は以下のようになる (詳細はマニュアル参照[1])。

```
killval 9 cid
```

現状で考慮している文字列は "TERM", "KILL", "QUIT", "HUP", "INT" の 5 つである。

[1] kill(2), signal(3) 等

5.3 表 示

print は数字や文字列、変数内容を表示する構文である。

`print` <*expr*> `print`

```
print 34
print 4+9
print "helo"
print a
print theserver.netif[1].ipaddr
```

5.4 実 行 状 態 変 更

`exit` `exit`

`exit` <*val*>

評価器の評価が終了する。値 val を与えた場合は、その値をプロセスの終了値とする。

`abort` `abort`

評価器が異常終了する。

`atexit` <*block*> `atexit`

終了時の処理をあらかじめ登録する。
以下のような記述では、32 が表示された後に文字列 bye が表示される。

```
atexit {
    print "bye"
}
print 32
```

`noatexit` `noatexit`

登録した終了時の処理を取り消す。

`sleep` <*second*> `sleep`

`msleep` <*milsecond*> `msleep`

指定された時間だけスリープ (何もしない) する。明示的な待機やタイミング調整で用いられる。ただし、厳密な精度は期待できない。

5.5 ディレクトリ

作業ディレクトリを移動する。

`chdir` <*"path"*> `chdir`

5.6 メッセージ交換

実験全体を管理するマスタ、個々のノードで実験を駆動するスレーブ間でメッセージを交換することができる。このメッセージ交換により実験進行の信号や個別処理終了の応答の記述が可能となる。

5.6.1 メッセージ送信

send	send [*node-name*] <*"message"*>

ノードへメッセージを送信する。ノード名 node-name が省略された場合は、既定値のノードへ送る。スレーブではマスターが既定値となっている。

multisend	multisend <*nodeset-name*> <*"message"*>

ノード集合のメンバへメッセージを送信する。多数ノードへのメッセージ送信を容易にするために設けられている。

意味上は一斉に送信するが、直列処理になるため集合内の最初と末尾で少々の遅延がある。数十ノードに対して数十 ms 程度。

5.6.2 メッセージ受信

メッセージを受信し、受信内容を変数に格納する。

recv	recv [*node-name*] <*var*>

ノードへメッセージを受信する。node-name が省略された場合は、既定値のノードから読み込む。スレーブではマスターが既定値となっている。

recv では受信内容を一つの要素 (数字や文字列) として扱う。複数の要素を受信する場合は、以下の recva を用いる。recva は受信内容を配列として変数に格納する。

recva	recva [*node-name*] <*var*>

5.6.3 メッセージ待機

sync	sync <*block*>

sync は処理を同期するための構文で、ブロック内に処理を再開する条件を列挙する。ブロック内の条件が満たされるまで休止する。ブロック内の条件には msgmatch および timeout を記述できる。

```
sync {
    msgmatch foo "done"
    timeout 900
}
```

msgmatch	msgmatch [*node-name*] <*"message"*>

node-name から指定メッセージを受信を待つ。

multimsgmatch	multimsgmatch <*nodeset-name*> <*"message"*>

nodeset-name で指定されたノード集合からの受信を待つ。ノード集合メンバ全員からの受信内容が揃うまで待つ。

timeout	timeout <*seconds*>

タイムアウト時間の指定。

5.7 制 御 構 造

`if (`<*cond*>`) `<*block*> `if`

`if (`<*cond*>`) `<*block1*>` else `<*block2*>

条件を満たした場合に block に記述された処理を行う。else を加えて記述した場合は、条件を満たした場合に block1、満たさない場合に block2 を処理する。

```
if(i%2==0) {
    print "even"
}
else {
    print "odd"
}
```

`repeat `<*num*>` `<*block*> `repeat`

指定回数分処理を繰り返す。

```
repeat 10 {
    print "a"
}
```

`while(`<*cond*>`) `<*block*> `while`

条件を満たしている間、処理を繰り返す。

```
i=0
while(i<10) {
    print "a"
    i++
}
```

`for(`<*init*>`;`<*cond*>`;`<*incr*>`) `<*block*> `for`

初期処理 init のあと、条件 cond を満たしている間、処理 block を繰り返す。繰り返す際には処理 incr を行う。

```
for(i=0;i<10;i++) {
    print "a"
}
```

`loop `<*block*> `loop`

単純に処理を繰り返す。

```
i=0
loop {
    if(i>=10) {
        exit
    }
    print "a"
    i++
}
```

foreach　　foreach <*var*> <*array-name*> <*block*>

配列の要素分だけ処理を繰り返す。

```
nodeclass Ccl {
}
nodeset cl class Ccl num 3
foreach i cl {
    print i.name
}
```

上の例ではノード集合分だけ繰り返すので、以下のような実行結果を得るだろう。

```
cl-0
cl-1
cl-2
```

msgswitch　　msgswitch <*"string"*> <*block*>

msgswitch は値にしたがって処理を分岐する。msgswitch のブロック中にはさらに以下のようなブロックが記述できる。値が指定した文字列と一致した場合そのブロック中の処理が行われる。

```
"<string>" {
  ...
}
```

そこで、msgswitch を使って以下のような処理を記述できる。

```
loop {
    recv cmd
    msgswitch cmd {
      "start" {
        wakewait "/bin/thetarget" "/bin/thetarget" "start"
        send "start-ack"
      }
      "stop" {
        wakewait "/bin/thetarget" "/bin/thetarget" "stop"
        send "stop-ack"
      }
      "done" {
        sleep 30
        exit
      }
    }
}
```

5.8 インターフェース走査

`netiffit` <*"path"*> `netiffit`

指定されたパスのプログラムを使ってネットワークインターフェースを走査する。現在のところ、この機能は SpringOS に含まれている ifscan [2] を想定している。

netiffit 実行後はデバイス名が決まるため、ifconfig 等のネットワーク関連の設定が容易となる。

```
netiffit "/sim/ifscan"
wakewait "/sbin/ifconfig" "/sbin/ifconfig" self.netif[0].rname \
        self.netif[0].ipaddr
```

`savenodeinfo` <*node*> <*"path"*> `savenodeinfo`

指定されたノードの情報をファイルに保存する。

5.9 ファイル転送

`netget` <*"URL"*> <*"path"*> `netget`

netget はネットワークを介してファイルを取得する構文である。取得先を URL (リモートパス) で、取得ファイルの格納場所をパス (ローカルパス) で指定する。現状では FTP と HTTP に対応している。

```
netget "ftp://install:install@172.16.3.200/foo.tar.gz" "foo.tar.gz"
```

リモートパスのみに .gz がついている場合は、リモートファイルを圧縮ファイルとみなして、伸長しながらローカルパスに格納する。双方に .gz がついている場合は伸長処理は行わない。

リモート	ローカル	処理
-	-	伸長なし
.gz	-	伸長
.gz	.gz	伸長なし

`netput` <*"path"*> <*"URL"*> `netput`

ネットワークを介してファイルを送出する。対象ファイルをパスで、送出先を URL で指定する。そして、netput 同様、パスに従って圧縮を行う。

[2] パッケージ ifsetup に収録

5.10 関数定義

func

func <*name*> <*block*>

func <*name*> (<*arg*> [, ...]) <*block*>

引数がなくても良い。手続きを記述するために用いても良い。

```
func alpha {
    32+432
}
func beta(a,b) {
    a+b
}
func gamma(a) {
    print a
}

print alpha()
print beta(7,23)
gamma(42)
```

5.11 スレッド分岐

thread

thread <*func*>

```
func gonbei {
    ...
    exit
}

thread gonbei
```

5.12 デバッグ向け

`settrace` ＜*”target”*＞ `settrace`
`unsettrace` ＜*”target”*＞ `unsettrace`

指定した評価器のモジュールの追跡出力を設定/解除する。現在の所、制御できるモジュールは以下のようになっている。

モジュール識別子	摘要
addr	IP アドレス
conn	コネクション
ejump	エラージャンプ
enbc	バッファリング入出力
encd	ENCD 処理
engen	汎用的要素処理
errp	ERRP 処理
esqp	ESQP 処理
eval	評価
mst	各種構造体
ncdb	ノード候補データベース
net	ネットワーク
nio	ネットワーク I/O
node	ノード
nsetup	ノードセットアップ
parse	K 言語解析
sw	スイッチ制御
symtbl	シンボル表
syntax	文法
tftpdman	TFTP 向けディレクトリ制御
thand	時刻処理
wol	WoL (wake on LAN)

評価に関する追跡出力の制御は以下のように行う。

```
chdir "/"
settrace "eval"
wakewait "/bin/ls" "/bin/ls"
unsettrace "eval"
```

第6章　変　数

多くの言語と同様に K 言語では変数が扱える。そして、値を代入した際にその型が決まる。データ型については後述 (8 章) する。

6.1　宣 言 ・ 初 期 化

変数名はアルファベットか _ (アンダー・スコア; under score) で始まるアルファベットかアンダー・スコアか数字の文字の並びが使える。文字列や数字等の変数の型は代入時に動的に定まる。

特に宣言の必要はなく、通常は以下のように = で値を代入すれば初期化となり、宣言も兼ねる。

```
binpath = "/usr/local/bin"
iteral = 23
```

初出の変数には undef という特殊な値が入っている。使う前に必ず初期化すること。

`array` <*var*>[<*num*>]　　`array`

配列は宣言が必要で、array 構文を用いる。

```
array m[3]
m[0] = "a"
m[1] = 43
m[2] = m[1]*2
foreach i m {
    print i
}
```

配列の長さは関数 alength で得られる。上の例は以下のように書くこともできる。

```
array m[3]
m[0] = "a"
m[1] = 43
m[2] = m[1]*2
for(i=0;i<alength(m);i++) {
    print m[i]
}
```

`assure` <*var*> `=` <*expr*>　　`assure`

---

assure は初期化していない変数に代入を施す構文で、指定された変数が初期化されていない場合は、与えられた値 (右辺の評価結果) で初期化する。

例えば、以下のような記述ファイルを評価器に与えると結果は 56 が得られる。

```
assure foo = 43+13
print foo
```

評価器がこの記述を評価する前に変数 foo に値が代入されていれば、評価器は 56 の代入を行わない。

6.2　特 殊 変 数 self

self はシナリオを実行するノードを指す変数である。グローバルシナリオではマスターを、ノードシナリオではスレーブを指す。文脈に依存する点に注意。

以下のようにノードの持つネットワークインターフェイスの IP アドレスを使う際に重宝する。

```
wakewait "/sbin/ping" "/sbin/ping" "-i" self.netif[0].ipaddr target
```

6.3　ス コ ー プ

ノードシナリオではノードシナリオ内で宣言した変数しか扱えない。グローバルシナリオではグローバルシナリオに至るまでに宣言した変数が扱える。

たとえば、以下のような実験記述では、ノードクラス foo のシナリオは変数 y は扱えないのでエラーとなる。一方、グローバルシナリオでは変数 x, y, z が扱える。

```
x = "ishikawa"
y = "toyama"

nodeclass foo {
    scenario {
       wake "/bin/echo" "/bin/echo" y
    }
}

z = "fukui"

scenario {
    wake "/bin/echo" "/bin/echo" x y z
}
```

6.4　名 前 の 衝 突

K 言語にとって変数と関数の違いはない。また、変数は要素一つの配列である。したがって、変数、配列、関数に同じ名前を割り当てることはできない。

第7章　内 蔵 関 数

7.1 数 学 演 算

`rand(<num>)` `rand`

整数の乱数を得る。乱数は num より小さな整数である。

```
rand(8) ⇒ 6
rand(8) ⇒ 4
```

`srand(<num>)` `srand`

乱数の種を与える。毎回異なる乱数が欲しい場合は、時刻を種にすると良いだろう。

```
srand(time())
print rand(10)
```

7.2 アドレス演算

`haddr(<"IPaddress">)` `haddr`

IP アドレスの文字列から、ホストアドレスのみを取り出す。

```
haddr("12.34.56.78/24") ⇒ "12.34.56.78"
```

`naddr(<"IPaddress">)` `naddr`

IP アドレスの文字列から、ネットワークアドレスを取り出す。

```
naddr("12.34.56.78/24") ⇒ "12.34.56.0/24"
naddr("172.16.3.44/23") ⇒ "172.16.2.0/23"
```

`addradd(<"base">,<delta>)` `addradd`

ある IP アドレスに数を加えて新たな IP アドレスを合成する。

```
addradd("172.16.2.0/23", 300) ⇒ "172.16.3.44/23"
```

7.3 型 変 換

tostring　tostring(<*num*>)

数字から文字列を生成する。

```
tostring(10) ⇒ "10"
k = 37
tostring(k) ⇒ "37"
```

なお、文字列は + で連結できる。

```
"haku"+"san" ⇒ "hakusan"
```

toint　toint(<*"string"*>)

文字列を数字に変換する。

```
toint("13") ⇒ 13
d = 54
toint(d) ⇒ 54
```

7.4 そ の 他

alength　alength(<*array-name*>)

配列の長さを得る。

time　time()

時刻を得る。一般的な UNIX 向けの C 言語の関数から得ていて、1970 年 1 月 1 日からの秒数。閏秒に関しては考慮されていないようだ。

getpid　getpid()

評価器のプロセス識別子。

7.5 実 験 的 関 数

以下は深層にある内部関数を使った実験的な実装である。仕様変更の可能性があるので、積極的には使わないこと。

`list(`<*expr*>`[, `<*expr*>`])` `list`

リストを生成する。

```
list(a,b) ⇒ (a b)
```

`length(`<*expr*>`)` `length`

リストの長さを得る。

```
length(list(1,2,3,4)) ⇒ 4
```

`quote(`<*expr*>`)` `quote`

指定されたい引数を評価しない。

`eval(`<*expr*>`)` `eval`

指定された引数を評価する。
以下のサンプルを実行すると 7 と 3 の階乗として、5040 と 6 を得る。

```
func frac(n) {
    if(n<2) {
        1
    }
    else {
        n*frac(n-1)
    }
}

a = list(quote(frac),3)

print frac(7)
print eval(a)
```

第8章　デ　ー　タ　型

K 言語は、単純型は INT, STR の 2 種類、複雑型は NODE, NET, NEIIF, AGENT, ADDFILE の 5 種類を持つ。そして、複数の NEITF と ADDFILE を扱うための型として、MNETIF, MADDFILE がある。たとえば、a.netif の型は MNETIF で a.netif[3] の型は NETIF となる。

名称	摘要
INT	整数
STR	文字列
NODE	ノード
NET	ネットワーク
NEIIF	ネットワークインターフェイス
AGENT	エージェント、通信相手のスレーブを指す
ADDFILE	追加ファイル
MNETIF	ネットワークインターフェイス配列
MADDFILE	追加ファイル配列

複雑な型は以下のような属性を持つ。

type	attr	R/W	get	put	desc.
NODE	name	RW	STR	STR	名称
	rname	RW	-	-	資源名
	description	R	STR	-	摘要
	method	RW	STR	STR	起動方法 (HDD/memory)
	ostype	RW	-	-	OS 種
	disktype	RW	STR	STR	ディスク種 (IDE/SCSI)
	partition	RW	STR	STR	パーティション (1,2 or 5)
	diskimage	RW	STR	STR	ディスクイメージ
	kernel	RW	STR	STR	カーネル
	n-netif	-	-	-	ネットワークインターフェイス数
	netif	RW	MNETIF	?	ネットワークインターフェイス
	n-addfile	-	-	-	追加ファイル数
	addfile	RW	MADDFILE	?	追加ファイル
	scenario	RW	any	any	シナリオ
	agent	RW	AGENT	?	スレーブ情報
	conn	P	-	-	スレーブコネクション
	lastmsg	P	-	-	最終メッセージ
	issued	-	-	-	発行時間
NET	name	RW	STR	STR	名称
	description	R	STR	-	摘要
	media	RW	STR	STR	メディア種
	ipaddrrange	RW	STR	STR	IP アドレス範囲
	bandwith	RW	STR	STR	帯域
	agent	-	-	-	通信エージェント
	conn	-	-	-	コネクション
NETIF	name	RW	STR	STR	名称
	type	?	-	-	種別
	phyport	?	-	-	スイッチの物理ポート
	media	RW	STR	STR	メディア種
	ipaddr	RW	STR	STR	IP アドレス
	macaddr	RW	STR	STR	MAC アドレス
	description	R	STR	-	摘要
AGENT	hostname	RW	STR	STR	ホスト名
	ipaddr	RW	STR	STR	IP アドレス
	portname	RW	STR	STR	ポート名
	pgname	RW	STR	STR	プログラム名
	msg	R	-	-	メッセージ
	lastmsg	R	-	-	最終メッセージ
ADDFILE	name	RW	STR	STR	名称
	description	R	STR	-	摘要
	file	RW	STR	STR	ファイル
	dir	RW	STR	STR	作業ディレクトリ

第9章　ユ ー ザ 定 義 型

複雑な構造を扱うためにユーザ定義型を用意してある。これは C 言語の構造体 (structure) に相当する。

`struct` <*struct-name*> <*block*>	`struct`
`struct` <*struct-name*> <*var-name*>	

<block> を記述すると型の定義。変数名を与えた場合は、定義された型を使った変数の宣言。

```
nodeclass serverC {
        netif media fastethernet
        netif media gigabitethernet
}
nodeclass firewallC {
        netif media gigabitethernet
        netif media fastethernet
}
struct site {
        node server class serverC num 2
        node firewall class firewallC num 2
        integer asnum
}
struct garaxy {
        struct site front num 2
        struct site rear num 2
        integer mnum
}

struct garaxy andromeda
print andromeda.mnum
print andromeda.front[0].asnum
print andromeda.front[0].firewall[0].name
print andromeda.front[0].firewall[0].netif[0].media
```

9.1 ユ ー ザ 型 要 素

ユーザ型には整数、文字列、ノードを含めることができる。

`integer` <*name*> [*num* <*num*>]	`integer`
`string` <*name*> [*num* <*num*>]	`string`
`node` <*name*> [*num* <*num*>]	`node`

要素数は <num> で指摘できる。

第10章　実　践

この章では実験を行う際の作業や注意点について述べる。エディタなどファイルを記述するプログラムについては他の文献を参照のこと。

10.1　作　業　手　順

エディタなどファイル編集プログラムについては他の文献を参照のこと。この節章では実験操作の手順例を示す。

1) 施設に依存する項目を調査する
   - リソース・マネージャ (RM) の稼働場所 (IP アドレス、ポート番号)。そして、そのバージョンと用いるプロトコル。
   - Wake On LAN agent の稼働場所 (IP アドレス、ポート番号)。
   - PXE ブートのための TFTP サーバ の稼働場所 (IP アドレスのみ、ポート番号は固定)。
   - NI を内蔵している OS のディスクイメージとカーネルの名称と場所。
   - 実験に使うことができる IP アドレス領域。
   - 実験に使うことができる VLAN 番号領域。
   - FNCP (NI 向け HTTP リダイレクトサーバ) の稼働場所 (IP アドレス、ポート番号)。
2) 実験に登場するノード集合を役割毎に分割する。
3) 役割毎にノード・シナリオを決める。
4) 各ノードで使うディスク・イメージを作成する
   - OS や各種ソフトウェアをインストールする。
   - ディスク・イメージを FTP サーバに保存する。

   Kuroyuri のパッケージを含めて、スレーブが自動的に起動するように設定すること。
5) 実験記述ファイルを作成する。
   - 施設
     - rmanager
     - wolagent
     - tftpdir
     - nidiskimage, nikernel, nipxeloader
     - fncp
   - 実験
     - user
     - project

  - ipaddrrange
  - nodeclass
  - node/nodeset
  - netclass
  - net/netset
- シナリオ
  - scenario

6) NI のためのファイル群やノードにインストールする実験ノードのディスクイメージが、それぞれ TFTP サーバと FTP サーバに格納されている事を確認する。

7) 実行

多くの場合、役割によってシナリオが決まり、必要なネットワーク・インターフェイスが定まる。したがって、実験を役割の視点で考えて、ノード・クラスや集合を記述すると良いだろう。

10.2 ポータビリティ

施設に依存する記述を集めて別ファイルにすると、実験記述のポータビリティが高まる。rmanager, wolagent 等が施設に依存する構文である。

たとえば、二つの施設で稼働させる場合は、施設に依存しない記述を expr.sc に、施設 A, B に向けにはそれぞれ facilityA.sc, facilityB.sc を用意しておく。施設 A では以下のように実行する。

```
% ./master facilityA.sc expr.sc
```

一方、施設 B では以下のように実行する。

```
% ./master facilityB.sc expr.sc
```

例えば、StarBED では以下に近い設定になるだろう。

```
#
# facility oriented information
#
# for StarBED 2003, 2004.
#
rmanager ipaddr "172.16.3.101" port "1234"

wolagent ipaddr "172.16.1.101" port "5959" ipaddrrange "172.16.1.0/23"
wolagent ipaddr "172.16.3.101" port "5959" ipaddrrange "172.16.3.0/23"

fncp ipaddr "172.16.3.101"
tftpdman ipaddr "172.16.3.101"

tftpdir "/osbank"
nidiskimage "fs.PICOBSD.old"
nikernel "picokernel"
nipxeloader "pexboot"

swtype name "silaswa001" type "IOS"
swtype name "silaswb001" type "CATOS"
swtype name "silaswb002" type "CATOS"
swtype name "silaswb003" type "IronWare"
```

10.3 タイミング

分散プログラミングの常として、ネットワーク実験ではタイミングが重要となる。K 言語ではメッセージを交換しつつ実験を進行させることが可能なので、実行時間の分からない実験にも対応できたり、対話的なシナリオが書けたり、他の方法に比べいくぶん便利になっている。しかし、便利になっただけでタイミングの困難さを完全に取り除いたわけではない。タイミングには十分に注意して欲しい。

K 言語で処理がブロックするは受信である。受信は recv や recva、そして sync である。特に sync は複数の入力の状態を検査してビジー・ループに近い状態になるため、注意が必要である。処理が止まったように見える場合は recv や recva、そして sync に注目すると良い。

ノードの起動は OS や管理サーバの状態など多くの要素に依存するため、ノード・シナリオ開始時刻は正確には揃わない。同一仕様の PC を十台程使う場合は、sleep で数秒待機する程度で回避できる。さまざまな仕様の PC を用いたり、数百台規模の場合には、開始時刻が広がると予想されるため、ノード・シナリオで準備完了のメッセージを送信し、グローバル・シナリオで sync を使って同期すると良いだろう。

```
nodeclass fooC {
    ...
    scenario {
        sleep 30
        send "ready"
    }
}
nodeclass barC {
    ...
    scenario {
        sleep 30
        send "ready"
    }
}
nodeset foo class fooC num 32
nodeset bar class barC num 64

scenario {
    sync {
        multimsgmatch foo "ready"
        multimsgmatch bar "ready"
    }
    ...
}
```

シナリオ終了時刻も原則として揃わない。多くの場合は問題は起きないが、最後のメッセージ送信が相手に届く前にシナリオが終了してしまい、コネクションが開放されてメッセージが届かないという現象が起きやすい。ノード・シナリオ、グローバル・シナリオともに最後に sleep を書いておくと、トラブルが減る。

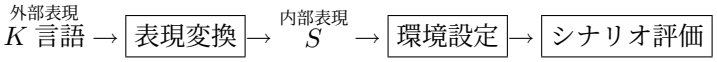
kuroyuri は自動的にスイッチを制御するため、K 言語では明示的なスイッチ制御の記述は不要である。ただし、スイッチ内部で設定内容が反映されるまでのタイムラグは検出できない。これは実際の通信をしてのみ確認できることで、自動的なスイッチ制御が原因ではない。シナリオ開始時に即座に実験対象プログラムを起動すると、このタイムラグで実験対象プログラムが不安定になったり、異常終了することがある。スイッチ制御が反映されるまで sleep でしばらく待つ (30 秒から 60 秒程度が目安) か、ping 等で疎通確認後に実験を開始すると良いだろう。

複雑な実験を記述すると、一方の送信回数が他方の受信回数より多かったり、あるいは、その逆のシナリオを書いてしまうことが起こる。送受信の回数が少ないシナリオを書くことをお勧めする。

タイミングチャートを描くと、すぐにミスが見付かることも多い。

付録A　言語処理系概要

K 言語は Kuroyuri で用いられる言語である。Kuroyuri は外部表現である K 言語を内部表現へ変換し、内部表現を評価する。



A.1　処理順序

基本的には記述順に処理される。atexit 文は例外的に最後に処理される。また、複数 atexit を書いても最後に記述した分のみ有効である。

グローバルシナリオを示す (node や nodeclass の属性でない) scenario 文が見付かると、スレーブを起動するために種々のノード設定の処理が始まる。逆に、scenario を書かなければ、ノード設定は行わない。

付録B　言 語 仕 様

K 言語が扱えるのは ASCII のみ、マルチバイトコードや日本語は扱えない。

B.1 文 法

```
program
    : lines
block
    : '{' '\n' lines '}' '\n'
    | '{' '\n' '}' '\n'
lines
    : lines '\n'
    | '\n' lines
    | lines line
    | line
    | error '\n'
line
    : RNOP wargs '\n'
    | RLOOP block
    | RREPEAT PINT block
    | RWHILE '(' expr ')' block
    | RFOR '(' expr ';' expr ';' expr ')' block
    | RFOREACH PSYM PSYM block
    | RIF '(' expr ')' block
    | RIF '(' expr ')' block RELSE block
    | RFUNC PSYM fargwrap block
    | RTHREAD PSYM '\n'
    | RARRAY PSYM '[' expr ']' '\n'
    | RSTRUCT PSYM structblock
    | RSTRUCT PSYM PSYM nodesetdecos
    | RNODE PSYM nodesetdecos '\n'
    | RNODESET PSYM nodesetdecos '\n'
    | RNODE PSYM nodeblock
    | RNODECLASS PSYM nodeblock
    | RSWEEPNODESET PSYM RVNTYPE PSYM '\n'
    | RNETSET PSYM netsetdecos '\n'
    | RBIND primary nodeblock
    | RBIND primary nodeline
    | RNETCLASS PSYM netblock
    | RSWTYPE swattributes '\n'
    | RRMANAGER agentattributes '\n'
    | RFNCP agentattributes '\n'
    | RENCD agentattributes '\n'
    | RTFTPDMAN agentattributes '\n'
    | RTFTPDIR PSTR '\n'
    | RNIDISKIMAGE PSTR '\n'
    | RNIKERNEL PSTR '\n'
    | RNIPXELOADER PSTR '\n'
    | RPXELOADER PSTR '\n'
    | RWOLAGENT wolagentattributes '\n'
    | RSETUPTIMEOUT setuptimeoutattributes '\n'
    | RSPARENODEMIN PINT '\n'
    | RSPARENODERATIO PINT '\n'
    | RRBHFILE PSTR '\n'
    | RSCENARIO block
    | RATEXIT block
    | RRECV expr '\n'
    | RRECVA expr '\n'
    | RRECV expr expr '\n'
    | RRECVA expr expr '\n'
```

```
    | RSEND expr '\n'
    | RSEND expr expr '\n'
    | RMULTISEND expr '\n'
    | RMULTISEND expr expr '\n'
    | RMSGSWITCH expr msgswblock
    | RSYNC syncblock
    | RPROJECT PSTR '\n'
    | RUSER PSTR PSTR PSTR '\n'
    | RUSER PSTR PSTR '\n'
    | RUSER PSTR '\n'
    | RIPADDRRANGE PSTR '\n'
    | RSAVENODEINFO pchains PSTR '\n'
    | RPRINT expr '\n'
    | RSLEEP expr '\n'
    | RMSLEEP expr '\n'
    | REXIT expr '\n'
    | REXIT '\n'
    | RABORT '\n'
    | RATTACH expr expr '\n'
    | RDETACH expr expr '\n'
    | RSWCONF '\n'
    | RNETIFFIT PSTR '\n'
    | RSETTRACE PSTR '\n'
    | RUNSETTRACE PSTR '\n'
    | R_DEBUGPRINT PSTR '\n'
    | RCD expr '\n'
    | RCHDIR expr '\n'
    | pchains '=' wakeline '\n'
    | wakeline '\n'
    | pchains '=' callline '\n'
    | callline '\n'
    | RKILL expr '\n'
    | RKILLVAL expr expr '\n'
    | expr '\n'
    | RASSURE PSYM '=' expr
structmemberdecos
    : /* empty */
    | structmemberdecos structmemberdeco
    | structmemberdeco
structmemberdeco
    : aclass
    | anum
nodesetdecos
    : /* empty */
    | nodesetdecos nodesetdeco
    | nodesetdeco
nodesetdeco
    : aclass
    | anum
    | aspare
netsetdecos
    : /* empty */
    | netsetdecos netsetdeco
    | netsetdeco
netsetdeco
    : aclass
    | anum
aclass
    : RCLASS PSYM
anum
    : RNUM expr
aspare
    : RSPARE expr
msgswblock
    : '{' '\n' msgcaselines '}' '\n'
    | '{' '\n' '}' '\n'
msgcaselines
    : msgcaselines msgcaseline
    | msgcaseline
msgcaseline
```

```
    : expr block
    | '\n'
syncblock
    : '{' '\n' synccaselines '}' '\n'
    | '{' '\n' '}' '\n'
synccaselines
    : synccaselines synccaseline
    | synccaseline
synccaseline
    : RMSGMATCH expr expr '\n'
    | RMULTIMSGMATCH expr expr '\n'
    | RMULTIMSGMATCH word expr expr '\n'
    | RTIMEOUT expr '\n'
    | '\n'
structblock
    : '{' '\n' structlines '}' '\n'
    | '{' '\n' '}' '\n'
structlines
    : structlines '\n'
    | '\n' structlines
    | structlines structline
    | structline
    | error '\n'
structline
    : RINTEGER PSYM structmemberdecos
    | RREAL PSYM structmemberdecos
    | RSTRING PSYM structmemberdecos
    | RSTRUCT PSYM PSYM nodesetdecos
    | RNODE PSYM nodesetdecos '\n'
    | RNODESET PSYM nodesetdecos '\n'
nodeblock
    : '{' '\n' nodelines '}' '\n'
    | '{' '\n' '}' '\n'
nodelines
    : nodelines nodeline
    | nodeline
nodeline
    : RNETIF PSTR netifattributes '\n'
    | RNETIF netifattributes '\n'
    | RAGENT agentattributes '\n'
    | RDISKIMAGE PSTR '\n'
    | RSCENARIO block
    | RMAXVNODES PINT '\n'
    | RVNTYPE PSYM '\n'
    | ROSTYPE PSTR '\n'
    | RMETHOD PSTR '\n'
    | RDISKTYPE PSTR '\n'
    | RKERNEL PSTR '\n'
    | RPARTITION PINT '\n'
    | RADDFILE PSTR addfileattributes '\n'
    | RADDFILE addfileattributes '\n'
    | '\n'
netblock
    : '{' '\n' netlines '}' '\n'
    | '{' '\n' '}' '\n'
netlines
    : netlines netline
    | netline
netline
    : amedia '\n'
    | abandwidth '\n'
    | aipaddrrange '\n'
    | '\n'
agentattributes
    : agentattributes agentattr
    | agentattr
agentattr
    : aipaddr
    | aport
    | atype
```

```
    | aprogram
    | ahost
    | aname
swattributes
    : swattributes swattr
    | swattr
swattr
    : aipaddr
    | aname
    | aport
    | atype
wolagentattributes
    : wolagentattributes wolagentattr
    | wolagentattr
wolagentattr
    : aipaddr
    | aport
    | aipaddrrange
setuptimeoutattributes
    : setuptimeoutattributes setuptimeoutattr
    | setuptimeoutattr
setuptimeoutattr
    : atotal
    | awarm
addfileattributes
    : addfileattributes addfileattr
    | addfileattr
addfileattr
    : adir
    | afile
netifattributes
    : netifattributes netifattr
    | netifattr
netifattr
    : amedia
    | abandwidth
    | aipaddr
    | amacaddr
    | anet
    | avia
aprogram
    : RPROGRAM PSTR
aname
    : RNAME PSTR
    | RNAME '\n'
atype
    : RTYPE PSTR
    | RTYPE '\n'
ahost
    : RHOST PSTR
    | RHOST '\n'
aipaddr
    : RIPADDR expr
    | RIPADDR '\n'
amacaddr
    : RMACADDR expr
    | RMACADDR '\n'
aport
    : RPORT expr
    | RPORT '\n'
amedia
    : RMEDIA PSYM
    | RMEDIA '\n'
anet
    : RNET PSYM
    | RNET expr
    | RNET '\n'
avia
    : RVIA PINT
    | RVIA '\n'
```

```
abandwidth
    : RBANDWIDTH expr
aipaddrrange
    : RIPADDRRANGE expr
    | RIPADDRRANGE '\n'
afile
    : RFILE PSTR
    | RFILE '\n'
adir
    : RDIR PSTR
    | RDIR '\n'
atotal
    : RTOTAL PINT
    | RTOTAL '\n'
awarm
    : RWARM PINT
    | RWARM '\n'
qexpr
    : pchains
    | '(' expr ')'
fargwrap
    :
    | '(' fargs ')'
fargs
    : fargs ',' expr
    | expr
wargs
    :
    | wargs qexpr
rdirs
    :
    | rdirs PLT qexpr
    | rdirs PGT qexpr
    | rdirs PGG qexpr
    | rdirs P2G qexpr
    | rdirs P2GG qexpr
    | rdirs P2J1
pexpr
    : qexpr
wakeline
    : wakes pexpr pexpr wargs rdirs
    | wakes pexpr pexpr
    | wakes pexpr pexpr wargs
wakes
    : RWAKEWAIT
    | RWAKE
callline
    : calls pexpr wargs rdirs
    | calls pexpr
    | calls pexpr wargs
calls
    : RCALL
    | RCALLW
expr
    : pchains '=' expr
    | expr PGT expr
    | expr PGE expr
    | expr PLT expr
    | expr PLE expr
    | expr PEQ expr
    | expr PNE expr
    | expr '+' '+'
    | expr '-' '-'
    | expr '+' expr
    | expr '-' expr
    | expr '*' expr
    | expr '/' expr
    | expr '%' expr
    | '(' expr ')'
    | '-' expr %prec UMINUS
```

```
    | pchains
list
    : '(' listv ')'
listv
    : expr
    | listv ',' expr
pchains
    : pchains '.' primary
    | pchains '.' primary '[' expr ']'
    | pchains '[' expr ']'
    | primary
bfunc
    : RTOSTRING
    | RTOINT
    | RRAND
    | RSRAND
    | RHADDR
    | RNADDR
    | RADDRADD
    | RGETPID
    | RTIME
    | RLASTMSG
    | RGETVNODES
    | RINSTANCE
    | RLENGTH
    | RALENGTH
    | RXPACK
    | RLIST
    | RQUOTE
    | REVAL
primary
    : bfunc list
    | bfunc '(' ')'
    | PSYM '(' fargs ')'
    | PSYM '(' ')'
    | word
    | const
word
    : RNETIF
    | RMEDIA
    | RROLE
    | RADDFILE
    | RNUM
    | RBANDWIDTH
    | RCPU
    | RDISK
    | RDISKIMAGE
    | RMEMORY
    | RANY
    | RALL
    | RAGENT
    | ROSTYPE
    | RIPADDR
    | RPORT
    | RIPADDRRANGE
    | RMACADDR
    | RMETHOD
    | RRNAME
    | RNAME
    | RFILE
    | RDIR
const
    : PSYM
    | PSTR
    | PINT
    | RDEFAULT
    | RT
    | RNIL
    | RUNDEF
    | PERR
```

B.2 予 約 語

以下のような単語が予約されている。変数や関数の名前には使えないので注意して欲しい。

ABORT, ADD, ADDFILE, ADDRADD, AGENT, ALENGTH, ALL, ALQ, ANY, APPEND, AQ, ARRAY, ASLISTQ, ASSURE, ATEXIT, ATOM, ATTACH, BANDWIDTH, BEGIN, BIND, BOUNDP, CALL, CALLW, CAR, CD, CDR, CHDIR, CLASS, COND, CONS, CPU, DEC, DEFAULT, DEFSTRUCT, DESCRIPTION, DETACH, DIE, DIR, DISK, DISKIMAGE, DISKTYPE, DIV, ELSE, ENCD, END, EQ, EQUAL, EVAL, EXIT, FILE, FNCP, FOR, FOREACH, FUNC, GE, GETAEQ, GETMAEQ, GETMEQ, GETPID, GETVNODES, GT, HADDR, HOST, IF, INC, INSTANCE, INTEGER, IPADDR, IPADDRRANGE, KERNEL, KILL, KILLA, KILLVAL, KILLVALA, LAMBDA, LASTMSG, LE, LENGTH, LET, LIST, LOOP, LT, MACADDR, MAKEAQ, MAXVNODES, MEDIA, MEMORY, METHOD, MKTHREAD, MOD, MSG, MSGMATCH, MSGSWITCH, MSLEEP, MUL, MULTIMSGMATCH, MULTISEND, NADDR, NAME, NCONC, NE, NET, NETCLASS, NETIF, NETIFFIT, NETMASK, NETSET, NIDISKIMAGE, NIKERNEL, NIL, NIPXELOADER, NODE, NODECLASS, NODESET, NOP, NOT, NUM, OSTYPE, PARTITION, PORT, PRINT, PRINTNET, PRINTNODE, PROGRAM, PROJECT, PUTAEQ, PUTMAEQ, PUTMEQ, PXELOADER, QUOTE, RAND, RBHFILE, REAL, RECV, RECVA, REPEAT, RMANAGER, RNAME, ROLE, SAVENODEINFO, SCENARIO, SEND, SET, SETALQ, SETAQ, SETQ, SETTRACE, SETUPTIMEOUT, SLEEP, SPARE, SPARENODEMIN, SPARENODERATIO, SRAND, STDERR, STDERRAPP, STDERRJOINSTDOUT, STDIN, STDOUT, STDOUTAPP, STRING, STRUCT, SUB, SWCONF, SWEEPNODESET, SWTYPE, SYNC, T, TFTPDIR, TFTPDMAN, THREAD, TIME, TIMEOUT, TOINT, TOSTRING, TOTAL, TYPE, UNDEF, UNSETTRACE, USER, VIA, VNTYPE, WAIT, WAKE, WAKEWAIT, WARM, WHILE, WOLAGENT, XPACK

付 録C　経 緯

StarBED チームははじめに実験環境記述の言語を設計した。その言語を受けて環境構築システム ConfigCoordinator が開発された。また、シナリオ記述言語が別に設計され、その評価システム SCE が開発された。Kuroyuri はこの二つのシステムを含んだ形で開発された。その言語 K が宣言文と実行文が混在した文法になっているのはこのためである。現在では、Kuroyuri とそれを補助するプログラム群を合わせて SpringOS と呼び、WWW サーバ (http://www.starbed.org/) で公開している。

Kuroyuri の名は北陸白山名物の黒百合にちなんだものである。

付録D　文 章 履 歴

[2004/06] WIDE deep space one BOF 向け資料で、phase1 を紹介する文章が登場。

[2004/06] HA 関連の項目を NI へ変更。施設関係 FNCP, SNCD, DMAN, WoL agent, switch に関する構文を導入。

[2004/07] nodegroup, netgroup を nodeset, netset に変更。

[2004/08] データ型、文法仕様を明示。

[2004/12] tostring, 文字列連結の + が登場。

[2005/11] mtk051103 に合わせて大幅書き換え。

- sncd が encd に変更
- 関数やスレッド, ユーザ定義型登場
- 未公開構文追記 (chdir 等)
- 索引スタイル変更

[2005/11] if–else, list, eval, kill 登場。タイミング追記。

[2005/11] 見出し類レイアウト変更。netget/netput 登場

索引