はじめてのLCONS

Ei-ji Nakama

Kanazawa.R#4

おしながき

リスト処理の言語

データを逐次に入れるのではなく各データの要素に次のデータのポインタをつけて記憶させる. IPL(Information Processing Language)[1] や, LISP[2] が起源.

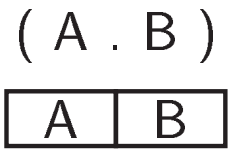
[1] 1956, A.Newell, J.C.Simon, H.Shaw
[2] 1960, J.McCarthy MITのグループ

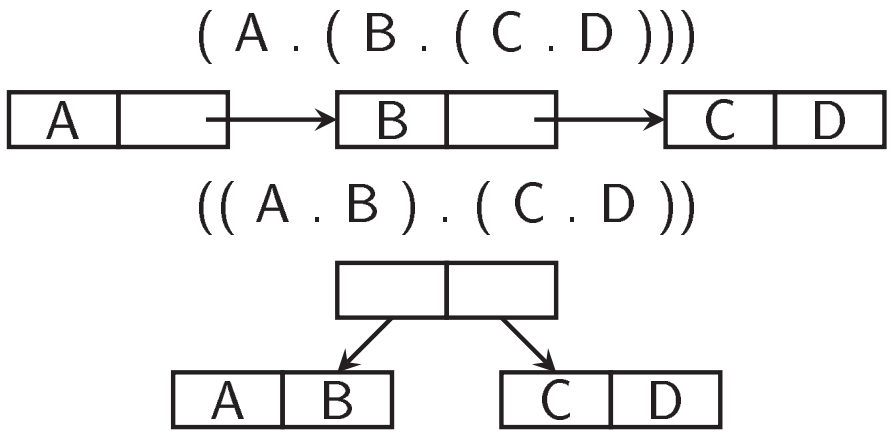
S式

S式の点表記法

$$\text{S式} ::= \text{アトム} \mid (\text{S式}.\text{S式})$$

(S式{左}.S式{右}) を図にするには２つに仕切った矩形に左右のS式を対応させる. S式がアトムならそこに書き, アトムでなければS式を表す矩形に矢印を引く.





S式 CAR部,CDR部,Nil

左側のポインタをCAR部,右側のポインタをCDR部とし, アトムNil[3]に特別な意味を持たせて以下のように表現するようになった

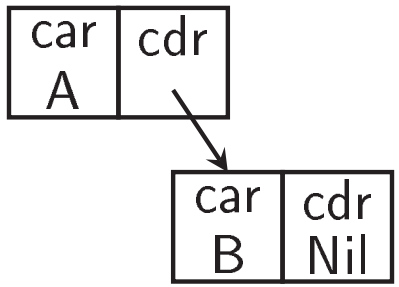
- (A) ≡ (A . Nil)
- (A B) ≡ (A . (B . Nil))
- (A B C) ≡ (A . (B . (C . Nil)))

点表記法では,括弧と.が多くなるのでSi^*がS式を表すとして, $(S1^*, S2^*, S3^*, ...Sn^*)$というリスト表記法[4]もある

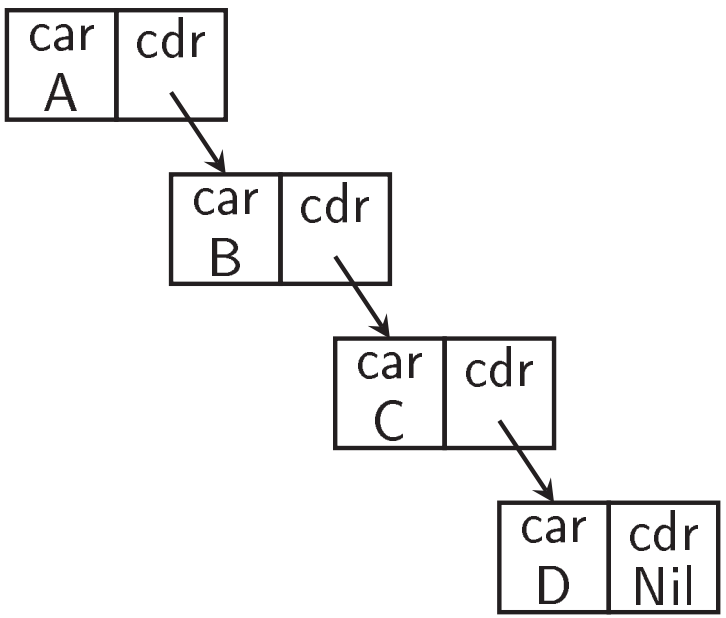
[3] latin語のNihilが由来
[4] Rにもlangnやlistn

S式 CAR部,CDR部,Nil 図

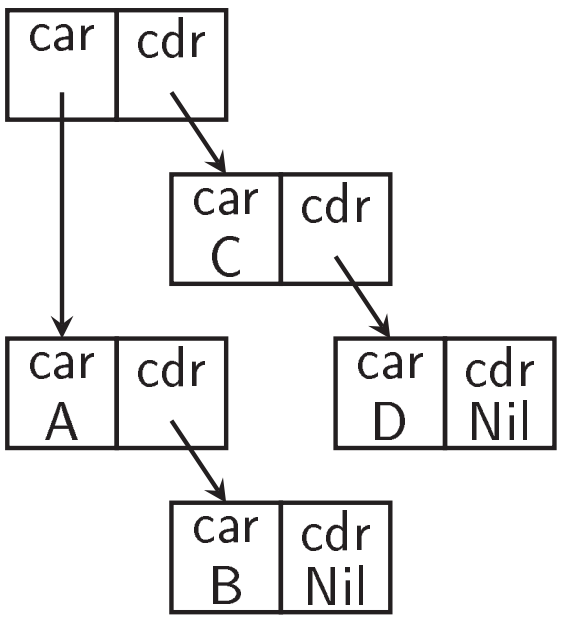
(A (B , Nil))



(A (B (C (D , Nil))))



((A (B , Nil)),(C (D , Nil)))



RでS式を作る1

計算式の作成

```
> sexp2 <- list( as.symbol("+"), 3, 1 )
> mode(sexp2) <- "call"
> sexp1 <- list( as.symbol("*"), 4, sexp2 )
> mode(sexp1) <- "call"
> sexp1
4 * (3 + 1)
```

RでS式を作る2

計算式の内容

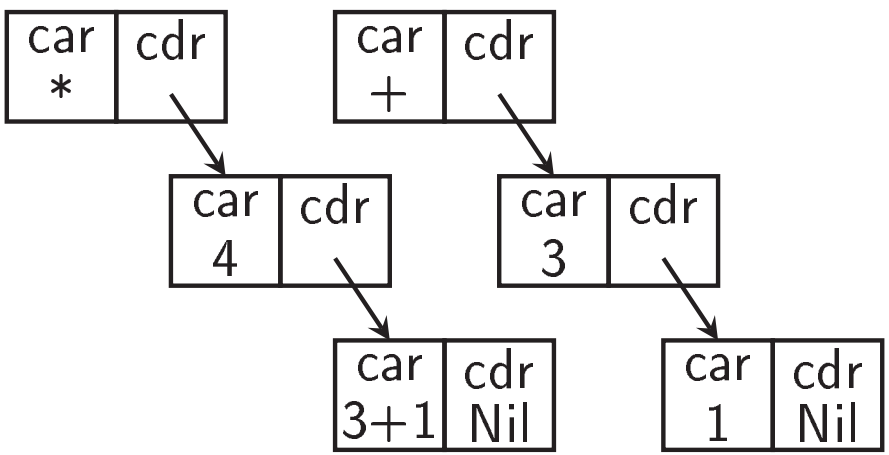
```
> as.list(sexp1)
[[1]]
`*`

[[2]]
[1] 4

[[3]]
3 + 1
```

```
> as.list(sexp1[[3]])
[[1]]
`+`

[[2]]
[1] 3

[[3]]
[1] 1
```

計算順序が木構造で確定されているので括弧(‘(’)が無い

RでS式を作る3

(* (4 (3 + 1 , Nil)))
(+ (3 (1 , Nil)))



RでS式を作る4

計算式の実行結果

```
> eval(sexp1)
[1] 16
```

LCONSを用いる

```
 1  #include <Rinterface.h>
 2  #include <Rinternals.h>
 3  #include <Rembedded.h>
 4  #include <R_ext/RStartup.h>
 5  SEXP Rf_deparse1(SEXP,bool,int);
 6
 7  int main(int argc, char **argv)
 8  {
 9      Rf_initEmbeddedR(argc, argv);
10
11      SEXP sexp2 = PROTECT( LCONS( install("+"), CONS(ScalarReal(3),
            CONS(ScalarReal(1), R_NilValue))));
12      SEXP sexp1 = PROTECT( LCONS( install("*"), CONS(ScalarReal(4),
            CONS(sexp2, R_NilValue))));
13
14      SEXP res1 = PROTECT( eval( sexp1, R_GlobalEnv ));
15      printf( "%g\n", REAL_ELT(res1, 0));
16
17      SEXP res2 = PROTECT(Rf_deparse1(sexp1, 0, 0));
18      const char *txt = CHAR(STRING_ELT(res2, 0));
19      printf("%s\n", txt);
20
21      UNPROTECT(4);
22
23      Rf_endEmbeddedR(0);
24      return 0;
25  }
```

LCONSのビルドと実行

Listing 1: ビルド

```
1  TARGET=lcons
2  export R_HOME=`R RHOME`
3  CC=`R CMD config CC`
4  CPPFLAGS=`R CMD config --cppflags`
5  LDFLAGS=`R CMD config --ldflags`
6
7  $CC $CPPFLAGS $TARGET.c -o $TARGET $LDFLAGS
8  ./$TARGET -q > $TARGET.stdout
```

Listing 2: 実行結果

```
1  16
2  4 * (3 + 1)
```

LCONSを用いる - リスト表記法

```
 1  #include <Rinterface.h>
 2  #include <Rinternals.h>
 3  #include <Rembedded.h>
 4  #include <R_ext/RStartup.h>
 5  SEXP Rf_deparse1(SEXP,bool,int);
 6
 7  int main(int argc, char **argv)
 8  {
 9      Rf_initEmbeddedR(argc, argv);
10
11      SEXP sexp2 = PROTECT( lang3( install("+"), ScalarReal(3),
            ScalarReal(1)));
12      SEXP sexp1 = PROTECT( lang3( install("*"), ScalarReal(4), sexp2)
            );
13
14      SEXP res1 = PROTECT( eval( sexp1, R_GlobalEnv ));
15      printf( "%g\n", REAL_ELT(res1, 0));
16
17      SEXP res2 = PROTECT(Rf_deparse1(sexp1, 0, 0));
18      const char *txt = CHAR(STRING_ELT(res2, 0));
19      printf("%s\n", txt);
20
21      UNPROTECT(4);
22
23      Rf_endEmbeddedR(0);
24      return 0;
25  }
```

LCONSのビルドと実行 - リスト表記法

Listing 3: ビルド

```
1  TARGET=lcons2
2  export R_HOME=`R RHOME`
3  CC=`R CMD config CC`
4  CPPFLAGS=`R CMD config --cppflags`
5  LDFLAGS=`R CMD config --ldflags`
6
7  $CC $CPPFLAGS $TARGET.c -o $TARGET $LDFLAGS
8  ./$TARGET -q > $TARGET.stdout
```

Listing 4: 実行結果

```
1  16
2  4 * (3 + 1)
```

Rのパーサーが解析した場合1

計算式の内容

```
> sexpa <- str2lang("4 * (3 + 1)")
> sexpa
4 * (3 + 1)
```

```
> as.list(sexpa)
[[1]]
`*`

[[2]]
[1] 4

[[3]]
(3 + 1)
```

```
> as.list(sexpa[[3]])
[[1]]
`(`

[[2]]
3 + 1
```

```
> as.list(sexpa[[3]][[2]])
[[1]]
`+`

[[2]]
[1] 3

[[3]]
[1] 1
```

Rのパーサーが解析した場合2

計算式の内容

```
> (sexpb <- str2lang("1:3 |> sqrt()"))
sqrt(1:3)
> as.list(sexpb)
[[1]]
sqrt

[[2]]
1:3
```

Fin

ご清聴ありがとうございました