Lego MindStorms 制御プログラムの対話型開発・実行環境

湯淺 太一（京都大学）†

わが国でもブロック玩具でよく知られている Lego 社が，MindStorms とよばれるロボットシステムを提供している．このシステムは，8 ビットの MPU を備えた RCX と呼ばれる「ブロック」にプログラムを与え，Lego シリーズのブロックを組み合わせることによって，さまざまなロボットを構築することができる．従来は，RCX の制御プログラムを記述するために，低学年の子供でも容易に使える言語あるいは本格的なプログラミングも可能な C 言語などの言語が提供されていた．前者はシンプル過ぎて，きわめて単純なプログラムしか記述できず，後者は子供も含めプログラミングの初心者には使えない上に，コンパイラ指向の言語であり，簡単な制御テストをするにも，完結したプログラムを作成し，それをコンパイルして RCX にダウンロードするという面倒な手間を必要とする．本発表では，低学年の子供やプログラミングの初心者でも容易に利用でき，相当高度な制御プログラム開発にも利用でき，しかも対話的にプログラムを開発できるプログラミング環境を報告する．このシステムは XS とよばれ，対話的プログラム開発が可能なプログラミング言語 Scheme をベースとし，それに MindStorms を制御するための諸機能を追加した言語を提供し，その言語で記述されたプログラムを対話的に開発するための Linux および Windows PC 上のプログラム開発・実行環境である．

1. MindStorms の概要

MindStroms[1] の中心となるのは，RCX とよばれるプログラム可能なブロック（縦 95mm ×横 65mm ×高さ 35mm）である．このブロックにモータやセンサなどの各種デバイスを装着し，他のブロックを組み合わせることによって，さまざまなロボットを自作できる (図 1参照)．

RCX の MPU は，16 ビットのアドレス空間を持ち 16 MHz で動作する日立 H8 プロセッサである．個々の RCX は，32K（あるいは 16K）バイトの ROM と 32K バイトの RAM を搭載している．ROM の内容は固定されており，RAM の一部は ROM プログラム実行のために予約されている．残りの RAM 空間を使って，RCX 用のファームウェア（あるいは OS）とユーザプログラムが動作する．二次記憶装置は備えていない．メモリの制約がこのように厳しいために，MindStorms 用のプログラミング環境を構築する際には，実行速度よりもメモリ使用量が重要となる．

RCX ブロックの上面には，5 文字程度を表示できる LCD ディスプレイ，四つの操作ボタン（電源の On-Off，プログラムを選択する Prgm，実行を開始する Run，実行状態をモニタする View の各ボタン），モータ等を接続する三つの effector ポート（ポート A〜C），センサを接続する三つのセンサポート（ポート 1〜3）が備わっている．RCX 自体は，ブロックの下半分に収納された 6 本の単三乾電池で動作する．

RCX は，フロントエンドである PC と赤外線を使っ

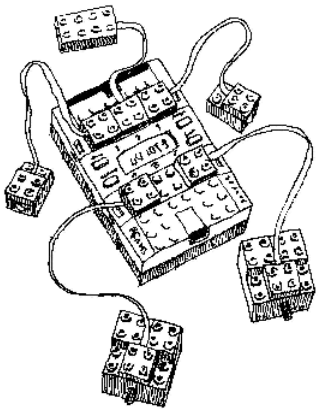

図 1 RCX の概観（モータ 2 個がポート A と C に，タッチセンサ 2 個がポート 1 と 3 に，光センサ 1 個がポート 2 に接続されている）．

† Interactive Programming Environment for Lego MindStorms by Taiichi Yuasa (Kyoto University).

本研究は，情報処理振興事業協会（IPA）から，平成 15 年度未踏ソフトウェア創造事業として支援を受けている．

て通信する．通信するためには，赤外線タワーとよばれる専用デバイスを PC に接続する必要がある．RCX には三つのモデルがあり，モデル 1.0 と 1.5 の赤外線タワーは PC のシリアルポートに，モデル 2.0 の場合は USB ポートに接続する．赤外線による通信は有効範囲が狭く指向性が強いために，PC と RCX はしばしば（特に，RCX が移動するロボットに組み込まれている場合に）メッセージをとりこぼす．

2. システムの概要

上述のように，RCX のメモリ制約は厳しく，プログラミング環境の中核となる実行系はきわめて小さくなければならない．このために，本システムを XS (eXtra Small) とよんでいる．この XS システムの主要な特徴を以下にあげる．

- 対話型プログラム開発環境を提供する．
  - 通常の Lisp 処理系と類似の，read-eval-print ループを持つ．
  - 対話的に関数を定義し，再定義できる．
  - エラーが検出されたときは，適切なエラーメッセージとバックトレースを表示する．
  - 関数呼び出しの引数と戻り値を表示する関数トレース機能を提供する．
- 実行系は RCX 内で自律的に動作する．
  - データオブジェクトを動的に割り付け，ごみ集めする．
  - 真に末尾再帰的なインタープリタによってプログラムを実行する．
  - プログラムエラーや，スタック／バッファオーバーフローに対して頑健である．
  - 端末機割り込み機能を持つ．
- ロボット制御のための十分な機能を提供する．
  - Scheme[2] をベースにした言語をサポートする．ただし，Scheme の一級継続は含まないので，この言語を Scheme ではなく，Lisp の一方言とみなしている．
  - モータ，センサ，ランプなどの Lego デバイスのためのインターフェイスを提供する．
  - イベント待ち，タイマ，非同期イベント割り込み（後述）の機能を提供する．

XS のシステム構成を図 2に示す．ユーザは，フロントエンドである PC（Linux または Windows）を使ってプログラムを開発する．ユーザが Lisp の S 式を一つ入力すると，フロントエンドシステムの reader がこれを読み込み，簡単な前処理を行ったあと，RCX 側の実行系（evaluator）に送信する．実行系はこの

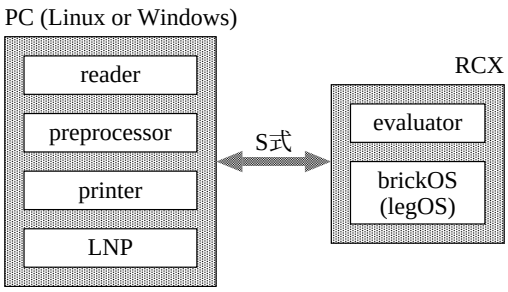

図 2　XS システム構成．

S 式を評価して結果をフロントエンドに返送する．フロントエンドシステムの printer がこの結果を PC のディスプレイに表示する．この一連のステップを繰り返すことによって，ユーザは MindStorms 用のプログラムを開発するのである．この繰り返しは，通常の Lisp 処理系における read-eval-print ループに相当する．通常の処理系と本質的に異なるのは，実行系が別プロセッサ上で動作し，赤外線通信を使って S 式を受け渡す点である．

実行系は，RCX 用に開発されたオープンソースの超小型 OS である brickOS[3] 上で動作する．brickOS はかつて legOS[4] とよばれていた OS の後継である．後述のように，開発当初の XS は legOS を使用していたが，現在は brickOS に移行している．RCX とフロントエンド PC との間の通信には，brickOS の一部として配布されている LNP (LegOS Network Protocol) を使用している．

3. データ型

XS は，次のデータ型を提供する．

- 真偽値（`#t` と `#f`）
- 14 ビット符号付き整数
- 空リスト
- コンス・セル
- 関数
  - 組み込み関数
  - 関数閉包（ユーザ定義関数）
- 記号
  - 組み込み記号（組み込み関数の名前）
  - ユーザ定義記号

RCX の MPU では，1 ワードは 16 ビット長であり，整数も 16 ビット長である．しかし後述のように，実行系ではワードの下位 2 ビットを，オブジェクトの型を区別するためのタグとして使用しているので，XS で使用できる整数は 14 ビット長となっている．

実際のプログラムでは，これらの型の他に，PC ディ

スプレイあるいは RCX の LCD ディスプレイにメッセージを表示するために，文字列と文字をデータとして扱いたい．XS では，厳しいメモリ制約のために，これらを一級オブジェクトとしては実装していない．そのかわり，フロントエンドの reader が文字コードのリスト（文字列の場合）および文字コード（文字の場合）に展開する「疑似データ型」として実装している．例えば，入力された文字列`"abc"`は `(97 98 99)` に変換され，文字 `#\a` は `97` と変換される．

さらに，プログラム記述を簡単にするために「reader 定数」というものを用意している．reader 定数は，Common Lisp[5] のキーワードと同様に，コロン（`:`）で始まる名前を持ち，入力時に reader によって整数に変換される．主な reader 定数を次にあげる．

- `:most-positive-integer`, `:most-negative-integer`
- `:a`, `:b`, `:c`（effector ポート指定）
- `:off`, `:forward`, `:back`, `:brake`, `:max-speed`（モータの動作指定）
- `:white`, `:black`（光センサ値指定）
- `:La0`, `:La#0`, `:Si0`, ..., `:So#8`, `:La8`（音階指定）

4. 組み込み関数

本稿末尾に，XS が提供する全関数（特殊形式を含む）をあげる．前半が通常の Lisp 処理系にもみられる一般的な関数群であり，後半が MindStorms 専用の関数群である．一般的な関数については，ベースとした Scheme の知識があれば，おおよその見当がつくであろう．ここでは，特に注意を要する点をあげるにとどめる．大域変数と大域関数を定義するには，トップレベル形式の `define` を使う．Scheme 同様，XS においても変数と関数は名前空間を共有する．制御関係では，Scheme の一級継続を提供しないかわりに，Common Lisp などで非局所的脱出のために用いられる catch&throw を提供している．繰り返しのための構文は特に用意せず，Scheme 同様に，関数の末尾再帰的呼び出しによって実現する．例えば，無限ループは次の式で実現できる．

```
(let loop () (loop))
```

乱数を生成する `random` は，ロボットにランダムな動作をさせたいときに有効なので採用した．文字列と文字は，reader がそれぞれ整数リストと整数に変換してしまうので，一般的な出力関数の `write` を使うと，整数リストあるいは整数として表示されてしまう．そのために，文字列と文字専用の関数をそれぞれ用意した．実行系は自動的にごみ集めを起動するが，強制的にごみ集めを行いたいときは `gc` を使う．`gc` は，利用可能なセル数を値として返す．

次に，MindStorms 専用の関数について簡単に説明する．トップレベル形式の `last-value` は，最後に実行した S 式の値を返す．通常，実行系が S 式を評価すると直ちにその値が PC ディスプレイに表示されるが，移動するロボットを制御している場合は，赤外線通信の制約のために値をフロントエンドに返送できないことがある．そのような場合に，赤外線通信が可能となってから `last-value` を使えば，いったんとりこぼした値を受け取ることができる．

`with-watchers` は，非同期イベント割り込みのための構文である．XS はメモリ制約のためにマルチスレッド機能を持っていない（スレッドごとに割り当てるスタック領域を確保できない）が，`with-watchers` を使えば，マルチスレッド機能に近い非同期動作を実現できる．その一般形は

(`with-watcher` ((*event*$_1$. *handler*$_1$) ... (*event*$_n$. *handler*$_n$)) . *body*)

である．この式が実行されると，本体である *body* の実行が開始される．本体の実行中，*event*$_1$〜*event*$_n$ の各条件が成り立つかどうかを，実行系がときどきチェックする．もしいずれかの条件 *event*$_i$ が成り立てば，本体の実行を一時的に中断し，対応する *handler*$_i$ を実行し，実行が終了すれば本体の実行を再開する．

`time` は，XS 起動後の経過時間を 1/10 秒単位で整数として返す．XS の整数は 14 ビット長なので，`time` の返す値は，約 13 分でオーバーフローしてしまう．このために，`reset-time` を使ってシステムクロックを 0 に初期化しなおしてから時間を計測のが賢明である．

XS で利用可能なセンサは，光センサ，角度センサ，温度センサ，タッチセンサの 4 種類である．このうち光センサと角度センサはアクティブセンサであり，センサを使用する前に activate しておく必要がある．また使用後はバッテリの消耗を押えるために inactivate するのが望ましい．`light` は，指定されたセンサポートに接続されている光センサが感知した光量を 0〜98 の整数として返す．`rotation` は，activate されてからの角度の変動を 360/16 度単位の整数（1 回転すると ±16）として返す．`temperature` は，温度センサが感知している現在温度を整数（摂氏）として返し，`touched?`は，タッチセンサが物体に接触しているかどうかの真偽値を返す．

モータを制御するためには，まず `speed` で速度を

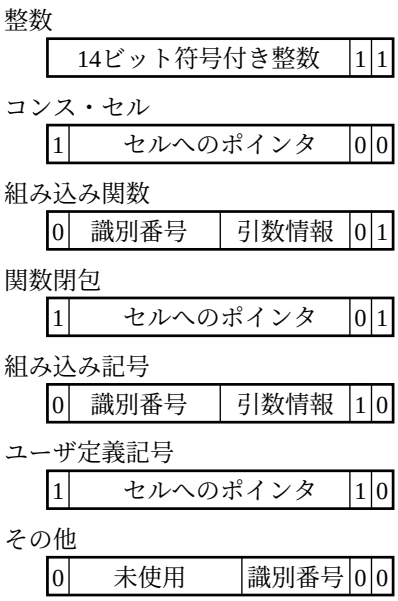

図 3　データ表現.

指定しておいてから，`motor` を使って動作（オフ，前進，後退，停止）を指定する．前進か後退かは，RCX の effector ポートへの接続の向きに依存するので，意図どおりの動作をしない場合は，接続の向きを変更する．RCX はシンプルな音源を備えており，簡単な曲を演奏することができる．曲は音程と音の長さを連想リストの形で `play` に与える．`play` は演奏を開始すると，終了を待たずに直ちにリターンする．直前の演奏が終了したかどうかを調べるには，`playing?`を使う．

XS では，RCX ブロックについている四つのボタンのうち Prgm ボタンを，プログラムで利用可能な信号を送る手段として利用できる．例えば，プログラムの起動後，特定の動作を開始するタイミングを指定する場合に便利である．このためには，

```
(wait-until (pressed?))
```

とすればよい．

LCD ディスプレイを操作する関数には，文字列（実際には文字コードのリスト）を表示する `puts`，特定の位置に文字（実際には文字コード）を表示する `putc`，LCD ディスプレイ全体をクリアする `cls` がある．`battery` は RCX のバッテリー残量を調べる関数である．バッテリーは赤外線通信にも使われるので，電圧が低下すると，フロントエンドとの交信もできなくなる．そこで，この関数をときどき使って，電圧が低ければ適宜バッテリーを交換することが望ましい．`linked?`は，RCX がフロントエンドと交信可能かどうかを調べる関数で，移動するロボットが交信できなくなった場合に，交信を再開するために欠かせない．

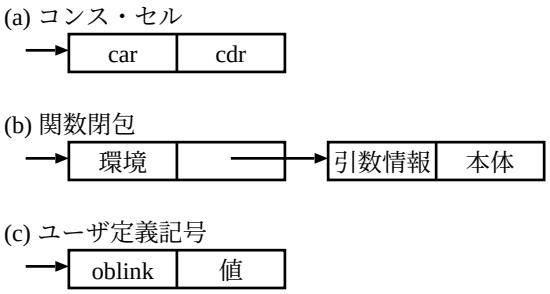

図 4　セルの解釈.

5. 実行系の概要

図 3に，XS 実行系が採用しているデータ表現を示す．MPU の 1 ワードは 16 ビットであり，その下位 2 ビットをデータタグとして使用している．整数，組み込み関数，組み込み記号，その他の真偽値や空リストは即値であり，コンス・セル，関数閉包，ユーザ定義記号は，ヒープに割り付けられたセルへのポインタを記憶している．小さいヒープ領域を有効に利用するために，ヒープには 2 ワードのセルだけを配置し，コンス・セル，関数閉包，ユーザ定義記号の実体は，これらのセルを組み合わせることによって表現している．それぞれのセルがどのように解釈されるかは，それがどのオブジェクトからリンクされているかに依存する（図 4参照）．

ヒープに割り付けられるセルがすべて同じサイズなので，ごみ集めに（GC）は，単純なマーク・スイープ GC を採用し，利用可能なセルはフリーリストとして管理している．実時間性が要求されるロボットを制御するためには，一般的には実時間 GC が望ましいが，ヒープが小さいことと，RCX の MPU が比較的高速であることから，XS の場合はプログラムを中断する停止型 GC でも十分機能するようである．実際，これまでの RCX 使用経験では，GC の停止によるトラブルは報告されていない．

XS のシステム全体は C 言語で記述されている．RCX 上で動作する実行系は，RCX の H8 MPU 用の GNU クロスコンパイラを使ってコンパイルする．フロントエンドシステムのほうは，通常の C コンパイラを利用する．Linux 版は GNU コンパイラを，Windows 版は Visual C を使っている．実行系を C 言語で実装したために，真に末尾再帰的なインタープリタを実現するために，新しい実装技術を開発する必要があった．これについては，本稿の範囲外なので，他の機会に報告したい．

現時点でのメモリ使用量を以下に示す.

- brickOS: 9K バイト
- 実行系バイナリ: 11K バイト（全組み込み関数を含む）
- 赤外線通信用入出力バッファ: 256 バイト
- C 言語スタック: 1K バイト（= 512 ワード）
- 変数スタック: 0.5K バイト（= 256 ワード）
- ヒープ: 5K バイト（1280 セル）

ヒープサイズは，実行系を RCX にダウンロードした後で，残っている利用可能メモリ量を割出し，きりのよい数値に丸めて決定している．メモリ使用量の合計が RAM 容量の 32K バイトに達しないのは，RAM 領域の一部が ROM プログラム実行用に予約されているためである．

6. 現状と今後

シリアルタワー（PC のシリアルポートに接続する赤外線タワー）を利用する RCX モデル 1.0 および 1.5 に対しては，Linux 版と Windows 版ともにプロトタイプが完成しており，Redhat Linux，Debian Linux，Windows XP 上で動作が確認されている．USB タワーを利用するモデル 2.0 に対しては，現状では，Windows 版は対応しているが，Linux 版はまだ対応していない．これは，ベースとなる brickOS が，Linux 上の USB タワーをサポートしていないためである．いずれ機会をみて実装したい．

XS のプロジェクトは，まず本稿の筆者が，シリアルタワー対応の Linux 版プロトタイプを legOS 上に開発することから始まった．開発開始時に legOS の後継である brickOS がすでに存在したが，legOS と比較して brickOS は規模が大きいために，より多くのメモリ領域を利用できる legOS を採用した．筆者には Windows のプログラミング経験がとぼしいので，Linux 版プロトタイプが完成した時点で，Windows 上への移植を米国フランツ社に委託した．当時，Cygwin 対応の legOS が存在したが，Windows 上では直接動作していなかった．このためにフランツ社では，Windows 対応の brickOS を使用し，そこから XS に不要な機能（マルチスレッド機能など）を削除することによって，当初の legOS よりもコンパクトな brickOS を構築し，適宜修正を加えることによって XS を動作させることに成功した．移植作業の中心となった John Foderaro 氏にこの場を借りて感謝したい．

XS は，オープンソースのソフトウェアとして，次の Web ページから公開する予定である．

`http:\\www.xslisp.com`

本稿執筆時点では，リファランスマニュアルや発表資料などのドキュメント類が，この Web ページからダウンロード可能である．プログラミングシンポジウム当日までには，ソフトウェアがダウンロードできるよう努力する．

参 考 文 献

1) LEGO.com MindStorms Home, `http://mindstorms.lego.com/`.
2) IEEE Standard for the Scheme Programming Language, IEEE (1991).
3) brickOS at SourceForge, `http://brickos.sourceforge.net/`.
4) legOS Alternative Lego OS, `http://sourceforge.net/projects/legos`.
5) Guy Steele: *Common Lisp the Language, Second Edition*, Digital Press (1990).

付　　録

A.1 関数一覧

XS が提供する全関数（特殊形式を含む）を，それらの関数プロファイルとともに以下に列挙する．次の記法を用いる．

X*: 0 個以上の X
X^+: 1 個以上の X
$\{X_1|\cdots|X_n\}$: $X_1,\ldots,X_n$ のいずれか
[X]: X が省略可能

関数プロファイル $X_1, X_2, \ldots, X_n$ が関数名を除いて同一のときは，まず X_1 を書いて，そのあとに $X_2,\ldots,X_n$ の関数名を書くことがある．例えば，`or` の関数プロファイルは (`or` *expr**) である．セミコロン（;）以降はコメントである．

A.1.1 一般的な関数

- トップレベル形式
  (`define` *sym expr*)
  (`define` (*sym sym** [. *sym*]) *expr**)
  (`load` *file-name*) ; ファイルからのロード
  (`trace` *function-name*) と `untrace`
  (`bye`) ; セッション終了
- 基本的な形式
  (`quote` *object*)
  (`set!` *sym expr*)
  (`lambda` (*sym** [. *sym*]) *expr**)
- 制御
  (`begin` *expr**)
  (`apply` *function object** *list*)
  (`if` *expr expr* [*expr*])

(catch *expr expr**)
(throw *object object*)

- 条件式
  (and *expr**) と or
  (not *object*)
- 束縛
  (let [*sym*] ((*sym expr*)*) *expr**)
  (let* ((*sym expr*)*) *expr**) と letrec
- 型述語
  (boolean? *object*) と integer?, null?, pair?, symbol?, function?
- 比較
  (eq? *object object*)
  (< int^{+}) と >, =, >=, <=
- 算術計算
  (+ *int**)
  (- *int int**)
  (* *int**)
  (/ *int int*) と remainder
  (logand *int int*) と logior, logxor, logshl, logshr
- 乱数
  (random *int*)
- リスト処理
  (car *pair*) と cdr
  (cons *object object*)
  (set-car! *pair object*) と set-cdr!
  (list *object**)
  (list* *object** *object*)
  (list-ref *list int*)
  (append [*list** *object*])
  (assoc *object a-list*)
  (member *object list*)
  (length *list*) と reverse
- PC 端末の入出力
  (read) と read-char, read-line
  (write *object*)
  (write-char *char*)
  (write-string *string*)
- ごみ集め
  (gc) ; 値はフリーセルの個数

A.1.2 MindStorms 固有の関数

- トップレベル形式
  (last-value) ; 最後の値は何だっけ？
- 制御
  (sleep *int*) ; 単位 1/10 秒
  (wait-until *expr*) ; イベント待ち
  (with-watcher ((*expr expr**)*) *expr**)
  ; 非同期イベント割り込み
- システムクロック
  (time) ; 単位 1/10 秒（13 分でオーバーフロー）
  (reset-time)
- 光センサ（引数の数値はポート番号）
  (light-on {1|2|3}) と light-off
  (light {1|2|3}) ; 値は 0（まっ黒）〜98（まっ白）
- 角度センサ（引数の数値はポート番号）
  (rotation-on {1|2|3}) と rotation-off
  (rotation {1|2|3}) ; 単位 360/16 度
- 温度センサ（引数の数値はポート番号）
  (temperature {1|2|3}) ; 単位は摂氏
- タッチセンサ（引数の数値はポート番号）
  (touched? {1|2|3})
- モータ（引数の:a, :b, :c はポート名）
  (motor {:a|:b|:c}
  {:off|:forward|:back|:brake})
  (speed {:a|:b|:c} *speed*) ; 値は 0〜255
- 音
  (play ((*pitch* . *length*)*))
  (playing?)
- Prgm ボタン
  (pressed?)
- LCD ディスプレイ
  (puts *string*)
  (putc *char column*)
  (cls)
- バッテリ残量
  (battery) ; 単位 1/10 ボルト
- 赤外線通信
  (linked?) ; 通信可能かどうか