情報処理学会論文誌　プログラミング　Vol.6　No.4　1–9 (Dec. 2013)

Mindstorms NXTを対象としたKonoha処理系のコンパクト化

志田 駿介[1,a)]　井出 真広[1]　倉光 君郎[1]

受付日 2013年4月12日, 採録日 2013年9月2日

概要：本研究の目的は静的型付けを特徴とするスクリプト言語 Konoha の処理系を Lego 社が開発したプログラム可能なロボットシステムである Mindstorms NXT 上に移植し，Konoha のスクリプトを動作させることである．Mindstorms NXT は ARM7 CPU と 64 kB RAM を搭載したロボットであり，Konoha 処理系とリアルタイム OS TOPPERS の使用するメモリの合計を 64 kB に抑える必要があった．我々は Mindstorms NXT 上で動作する Konoha 処理系を TinyKonoha と名付け，パーサや各ランタイムのコンパクト化を行った．最終的に TinyKonoha の使用したメモリ量はコード領域，スタック領域，ヒープ領域合わせて 32 kB 程度となり，VM 命令セットやオブジェクト構造の再設計を行った言語ランタイムを開発した．その結果，組み込みシステム技術協会が主催する ET ロボコンの公式コースを走行可能な規模のスクリプトを実行することが可能であった．本論文では TinyKonoha の設計，省メモリ化の工夫を詳細に報告し，メモリ使用量の評価とともに省メモリ環境のスクリプト処理系開発の知見をまとめる．

キーワード：スクリプト言語，省メモリ化

Downsizing Konoha Scripting Language Intended for Mindstorms NXT

Shunsuke Shida[1,a)]　Masahiro Ide[1]　Kimio Kuramitsu[1]

Received: April 12, 2013, Accepted: September 2, 2013

Abstract: We present a small-size scripting language interpreter named TinyKonoha, which is originally developed as statically typed scripting language Konoha, that can be executed on Mindstorms NXT. Mindstons NXT is a robot developed by Lego Inc. and has ARM7 CPU and 64 kB RAM. We set a goal to execute the script that can controls Mindstorns NXT and can read a value from sensors. Our language processor is based on static typing script language Konoha and we called it TinyKonoha. The memory usage of TinyKonoha gets to less than 32 kB including code segment, stack segment and heap segment and we developed a new language runtime. In the result, writing a script, which can run the course of ET Robot Contest hosted by JASA, becomes possible. In this paper, we present the design and the implementation of TinyKonoha, the method of simplifying the language processor and the actual memory usage of TinyKonoha.

Keywords: scripting language, memory saving

1. はじめに

スクリプト言語処理系は，プログラミングしやすさをともなって様々な実行環境や応用領域で利用が増えている．組み込み領域は，C/C++による開発が主体であったが，アプリケーション高度化にともない，スクリプト言語処理系への期待が高まっている．同時に，厳しいコンピューティング資源の制約があり，スクリプト言語処理系の実装者にはチャレンジングな課題となっている．

本論文では，JASA の ET ロボコン競技会[*1]の走行体として人気のロボット環境 Mindstorms NXT 上でスクリプト言語によるプログラミングを目指しており，我々は静的型

[1] 横浜国立大学
Yokohama National University, Yokohama, Kanagawa 240–8501, Japan
[a)] shunsuke.sida@gmail.com

[*1] ET ロボコン競技会（http://www.etrobo.jp/）

付け言語 Konoha をさらにコンパクト化した TinyKonoha の設計と実装を行った．Mindstorms NXT の特徴は，厳しい RAM メモリ制約である．ロボット制御に必要な RTOS（リアルタイム OS）を搭載した状態で，言語処理系が自由に利用できる RAM は 32 kB 程度となる．この上で搭載可能になるように，実行コードとランタイムで使用するメモリ双方の削減が必要となる．

我々は Konoha 処理系のメモリ利用量を分析し，Konoha のコンパイラ部分の分離とオブジェクトのコンパクト化，バイトコードサイズ削減の 3 つの方法に取り組んだ．これにより，TinyKonoha を用いた Mindstorms NXT の制御を行うスクリプトが十分に記述可能なことを確認した．

本論文の構成は以下のとおりである．2 章ではスクリプト言語処理系を搭載する対象機器となる Mindstorms NXT と関連するライブラリ群について説明する．3 章ではコンパクト化の対象となる静的型付け言語 Konoha の概要を述べる．4 章では TinyKonoha の設計と実装について述べる．5 章では TinyKonoha のメモリ使用量について説明し，6 章では関連研究を述べる．

2. Mindstorms NXT

Mindstorms NXT は Lego 社が開発したプログラム可能なロボットシステム（図 1）である．我々はこの上でロボットプログラムをスクリプトによって記述することを目指している．本章では，Mindstorms NXT のプログラム実行環境について述べる．

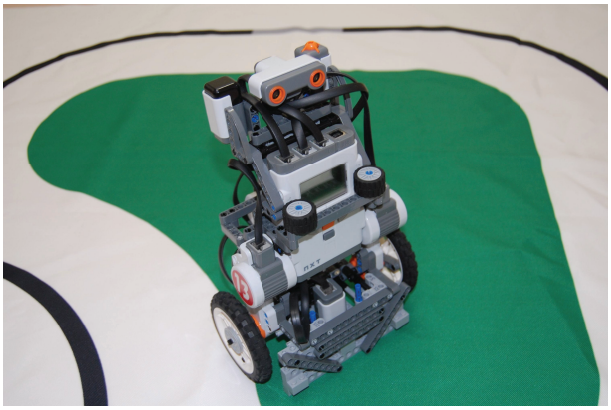

図 1　Mindstorms NXT
Fig. 1　Mindstorms NXT.

2.1 ハードウェア構成

Mindstorms NXT は，ユーザがプログラムしたロボットの制御プログラムを実行する計算機環境を持っている．基本的な計算機環境の仕様は次のとおりである．

- ARM7（32 bit/48 MHz）CPU
- 256 kB EEPROM
- 64 kB RAM

ユーザは，この計算機環境に対し，光センサやアクチュエータ，Bluetooth 通信機器など各種デバイスを接続し，プログラムから制御することができる．プログラムやデータは，ROM 領域に保存され，実行時に RAM 領域にロードされる．ただし，通常の組み込みシステムとは異なり，ROM 領域へのプログラム中からのアクセスは不可能である．したがって，ROM 領域にプログラムやデータの一部を配置できず，実行コード，静的データ，ヒープ，スタックを RAM 領域 64 kB 以内に抑える必要がある．

2.2 RTOS とソフトウェアスタックの構成

ロボット制御プログラムには，実時間処理に優れた RTOS（Real-Time Operating System）が必要である．たとえば，ロボットが立つのに必要な倒立振子制御には 4 ミリ秒の周期実行が必要であり，デッドラインミスは転倒につながる．

我々は，TRON ベースの RTOS である TOPPERS [1] を採用した．TOPPERS は，実行プロファイルが軽量であり，Mindstorms NXT 上での運用実績もあり，タスク管理，デバイスドライバ，倒立振子のライブラリなどが含まれている．

TOPPERS は，デバイスドライバを含めたバイナリのために Mindstorms NXT の RAM を 32 kB 使用する．

```
void mainLoop () {
  NXT.init();
    while (NXT.isRunning()) {
    NXT.updateStatus();
    if (NXT.getLightSensor() >= 600 /* threshold */) {
      NXT.control(50 /*speed*/,  50/*angle*/);
    } else {
      NXT.control(50 /*speed*/, -50/*angle*/);
    }
    NXT.wait(4); // wait 4ms
  }
}
```

図 2　ライントレーススクリプト
Fig. 2　A script of a line trace robot.

2.3　アプリケーション

我々は，後述するとおり Mindstorms NXT を用いてライントレース制御プログラムを実行することを目標としている．ライントレースプログラムとは，光センサからラインの位置を読み取り，その値をもとにアクチュエータを制御して，ラインの上を動作させるプログラムである．**図 2** に簡単なソースコードを掲載する．

Mindstorms NXT に接続された光センサの値を読み取り，得られた値をもとに旋回する向きを変更している．この動作を繰り返すことで，ラインに沿った Mindstorms NXT の制御を行うことができる．

3 章以降では，このようなライントレースプログラムをスクリプトで記述し，Mindstorms NXT 上で動作させることのできるスクリプト処理系への要求と実装について述べる．

3.　Konoha の概要

我々は上述の Mindstorms NXT 上に Konoha をベースとしてコンパクト化した TinyKonoha 処理系を載せることを目指している．本章では，簡単に Konoha の特徴を述べる．

3.1　言語の特徴

Konoha [2] は，C 言語スタイルの文法を持つ静的型付けオブジェクト指向言語である．Konoha は，オブジェクト指向プログラミング言語であるが，クラス定義などの文法はモジュールとして分離できるようになっている．Konoha にあらかじめ組み込まれる文法は以下のとおりとなっており，クラス定義などはライブラリをインポートすることで使用可能となっている．

- if-else 文，return 文
- メソッド定義，メソッド呼び出し
- Func 型と関数オブジェクト
- boolean 型と論理演算子
- int 型と四則演算子，比較演算子
- String 型と文字列連結演算子
- ローカル変数宣言

3.2　オブジェクトの構造と GC

Konoha は，すべての値をオブジェクトで表現するオブジェクト指向プログラミング言語である．オブジェクトは，ヒープ上でアロケートされたメモリで表現される．

Konoha のオブジェクトは，オブジェクトヘッダとフィールド（データ領域）からなる．オブジェクトのヘッダは，**図 3** で定義される構造を持つ．32 ビット環境化では，12 バイトになる．したがって，1 つの参照（値）だけを持つ最小オブジェクトのサイズは 16 バイトとなる．

オブジェクトのヘッダ領域にはクラス情報（classRef メンバ）とオブジェクトの状態を表す flags メンバが用意されている．Konoha でよく使われるオブジェクトのサイズは型ごとに異なり，最小の String 型で 32 バイト，Method 型で 64 バイトとなっている．なお，整数のみは，自動的に Unbox された値として表現される．

```
typedef struct {
  struct ObjectHeader {
    long flags;
    kClass *classRef;
    kPrototypeMap *m;
  } h; // オブジェクトのヘッダ情報
  struct kObject fields[5]; //Object body
} kObject;
```

図 3　Object 型のメモリレイアウト
Fig. 3　Memory layout of kObject.

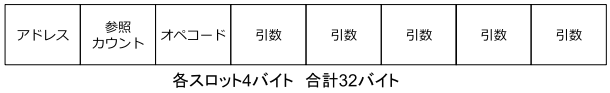

図 4　MiniVM のバイトコード
Fig. 4　Bytecode in MiniVM.

オブジェクトのメモリ管理は，自動的なメモリ管理機構 GC（Garbage Collection）を用いている．GC アルゴリズムには Ueno らの移動のないビットマップ世代別 GC アルゴリズム [3], [4] を実装している．オブジェクトはオブジェクトのサイズごとに用意されたセグメントで管理される．1 つのセグメントのサイズは，128 kB となっており，これを等分割して管理できるサイズのオブジェクトがビットマップ管理される．我々の実装では，セグメントの分割の単位は最小サイズの 32 バイトから，64, 128, ..., 1,024 バイトと設定されている．たとえば，48 バイトのオブジェクトは 64 バイトのオブジェクトとして管理される．世代の管理は，セグメントを管理するビットマップでされ，世代ごとにセグメントを分割する必要がない．Konoha 起動時に 128 個のセグメントが用意され，合計 16 MB が GC 管理のメモリ領域（ヒープ）となる．

3.3　MiniVM

Konoha は，**図 4** で示されたメモリレイアウトを持つバイトコードにコンパイルされ，独自の設計のバーチャルマシン（VM）型インタプリタ MiniVM の上で実行される．MiniVM は，Konoha のコードを実装するうえで必要な最小の命令セットを用いて実装したレジスタ型の VM である．MiniVM の命令セットは以下の 6 つで構成されている．

- レジスタ間の値の移動
- 定数の読み込み
- メソッド呼び出し
- メモリ割当て
- 分岐命令

```
struct KonohaStack {
  union {
    int i;
        float f;
  } unboxData;
  kObject *object;
```

図 5　Konoha スタック
Fig. 5　KonohaStack.

- 例外の発行

VM の高速化などで用いられる特化命令は用いず，加算，減算など数値演算を含む各種計算処理はメソッド呼び出しとして行う．命令セットは最小構成で設計を行っている一方で，命令長は（L2 キャッシュラインのサイズにあわせ）32 バイト固定となっている．Mini VM では，実行高速化のため，Direct Threaded Code [5] による命令ディスパッチの高速化を行っている．そのため，命令実行部のアドレスをポインタとして持っている．

Konoha/MiniVM では，メソッドの引数や返り値などメソッド間のデータの受け渡しには C スタックを直接利用せず，独自の Konoha スタックデータ構造（以下，Konoha スタックと呼ぶ）を用いる．特徴は，静的スクリプトの性質を生かして，整数値の自動的な Unboxing を行っており，数値とオブジェクトの参照を明確に区別可能なワイドスタックを採用している点である．

Konoha スタック上（**図 5**）で，メソッドを呼び出すときは，呼び出し元は引数と関数呼び出し後のスタックポインタのアドレス，呼び出した関数の情報を Konoha スタックに push し，対象のメソッドを呼び出す．返り値は，Konoha スタックから取り出すことができる．Konoha の関数呼び出し規約では，n 個の引数を持つ関数の呼び出しに対して，n + 5 個の Konoha スタックを使用する．

3.4　パーサ

Konoha は，スクリプト言語処理系としてスクリプトをそのまま実行するため，パーサ機能が内蔵されている．パーサは，Konoha オブジェクトを生成しながらコード生成を行う．まず，ソースコード（String）は，字句（Token）に分割され，それらが構文木（Node）に変換される．構文木からコード生成器によりインタプリタ向けのバイトコードが生成される．構文木を構成するこれらの要素は Konoha の GC が管理するオブジェクトとして定義されている．一方で，生成されたバイトコードは直接 malloc() でアロケートされた領域を用い，Konoha のオブジェクトとして管理していない．

3.5　メモリ消費量の見積り

Konoha のコンパクト化を試みる前にメモリ消費量の見積りを行う．Konoha は，Intel x86 アーキテクチャ上でビルドした場合，実行コードサイズは 131.8 kB であった．**表 1** に示したとおり，Konoha の実行コードの大部分がパーサとコード生成器で占められていることが分かる．ここで，C ライブラリはダイナミックリンクされて含まれていない．ただし，明らかに実行コードのみでも，Mindstorms NXT の RAM 容量を大きく超えている．

表 1　Konoha のバイナリサイズ
Table 1　Composition of Konoha's binary.

パーサとバイトコード生成器	110.6 kB
VM	7.4 kB
GC	13.8 kB
合計	131.8 kB

表 2　12 行のライントレーススクリプトの使用メモリ
Table 2　Memory usage of a script of 12 lines.

行数	12 行
ソースコードサイズ	191 バイト
構文木の使用オブジェクト数	161 個
構文木の使用メモリ	5.1 kB
バイトコードの命令数	24 個
バイトコードサイズ	768 バイト
Konoha スタックの使用スロット	17 個

実行時のメモリ利用量は，プログラムの規模に依存するが，ここでは 2.3 節で述べたライントレースプログラムを対象とした見積りを行った．**表 2** に，スクリプトのパース時，バイトコード生成後の実行時に使用するメモリ量を示す．コンパクト化以前の Konoha は Mindstorms NXT 上での動作とデバイスを制御する能力を持たないため，ライントレース制御に用いる API はスタブとして用意した．

表 2 に示したとおり，12 行のライントレース関数のコンパイル時には 161 個のオブジェクトで構成される構文木が生成されている．ここで，Konoha オブジェクトは 32 バイト以上必要となるため，少なくとも 5.1 kB のメモリが必要となる．一方この構文木から，24 命令からなるバイトコードが生成される．命令長は 32 バイト固定であるため，全体では 768 バイトメモリを使用していることになる．

3.4 節で述べたとおり Konoha の構文木は GC によって管理されているため，バイトコード生成後のオブジェクトは回収される．一方，バイトコードは Konoha の実行時につねに残り，回収されないことに注意する必要がある．以上の見積りにより，単純な省メモリ化のチューニングでは Konoha を Mindstorms NXT 上に搭載することは難しく，スクリプト処理系アーキテクチャの変更を含めたコンパクト化が必要であると結論付けられた．

4.　TinyKonoha の設計と実装

我々は，Mindstorms NXT 向けのコンパクト化した Konoha を TinyKonoha と名付けた．本章では，Konoha

のメモリ使用量の見積りを受けて TinyKonoha のコンパクト化の戦略をあげ，TinyKonoha 処理系におけるメモリ使用量削減の手法について述べる．

4.1　方針

TinyKonoha が利用可能なメモリ量は，2.2 節で述べたとおり，TOPPERS の実行のために 32 kB が必要であるため，64 kB 中たかだか 32 kB 程度である．前章で示したようにライントレーススクリプトの実行の際には，以下の順にメモリを使用していることが明らかとなっている．

1. 実行コード

表 1 に示したとおり Konoha はパーサ・バイトコード生成器の部分のバイナリサイズが 110 kB を超えており，この部分だけでも Mindstorms NXT 上で Konoha 処理系が使用可能な 32 kB を大幅に超えている．

2. ランタイムの使用メモリ

実行時に使用するオブジェクトの数とバイトコードの利用する領域はスクリプトによって大きく変化するが，これらの上限をより大きくするためにも，オブジェクトとバイトコードの構造体のコンパクト化は特に重要である．

この前提条件のもと，Konoha の各機能のコンパクト化の方針を述べる．

4.1.1　パーサ・バイトコード生成機能の分離

スクリプト言語処理系としてはパーサ・バイトコード生成器の部分は重要であるが，Mindstroms NXT に搭載するのは現実的とはいえない．しかし，スクリプトは外部の PC などで作成する必要があるため，パーサ機能は Mindstroms NXT の外部に置き，適切な連携機構をおくことで，スクリプト処理系として対話的な実行環境が構築できる．

一方，GC 部と VM 部は，21 kB 程度であり，32 kB のうちには搭載できる．ただし，スクリプトを実行するのに必要なヒープなどを確保するためにコンパクト化が必要となる．

我々は，GC，VM，連携機構をあわせたバイナリサイズを 12 kB 程度と目標値を定め，残り利用可能なメモリ 12 kB の内訳をヒープサイズ 4 kB，バイトコード領域 4 kB，Konoha スタック 4 kB 程度とした．このサイズに基づいて，さらなるコンパクト化の方針を検討した．

4.1.2　オブジェクトのコンパクト化

Konoha はプロトタイプデータへの参照など，今回の Mindstorm NXT 上の動作には直接必要ない実装が含まれている．これらのヘッダを削除し，さらに個別の String などよく利用されるオブジェクトのスリム化を行うことにした．

オブジェクトのコンパクト化は，GC のアロケータの構造を考えたサイズ決定が必要となる．今回は，固定サイズのオブジェクトに統一する方針を採用した．組み込みシステムの GC には，しばしばリアルタイム性が求められるが，今回はたかだか 4 kB 程度のヒープ領域であり，GC の最大停止時間も見積もりやすい．そのため，インクリメンタル GC などの GC アルゴリズムの採用を見送っている．

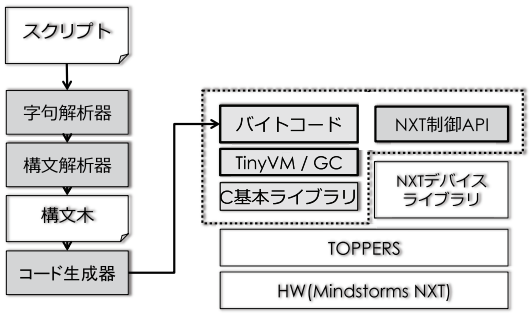

図 6　TinyKonoha の構成
Fig. 6　Design of TinyKonoha.

4.1.3　バイトコードサイズの削減

バイトコードサイズを減らすための方針としては，命令長の見直し，命令セットの見直しによって対応することができる．ただし，命令セットの見直しに関しては，文献 [6] などの文献を参考に，レジスタマシンのまま実装を進めている．

4.2　TinyKonoha の構成

我々は，TinyKonoha を**図 6** のとおりに構成することにした．まず，パーサ機能は Mindstorms NXT の外に置き，VM を Mindstormx NXT 上に置いている．TinyKonoha のスクリプトは外部環境でコンパイルされ，生成されたバイトコードは Bluetooth 通信によって送受信され，新たに追加したバイトコード・ローダによってロードされる．このため，バイトコードの受信待ち状態となった Mindstorms NXT に対して，外部の PC で編集されたスクリプトをコマンドライン上の操作のみで実行することが可能となる．

我々は，TinyKonoha 上の VM を TinyVM，GC を TinyGC と名付けた．Konoha の文法のうち，Mindstorms NXT の制御のために，最低限必要な TinyKonoha に組み込んだ．

- 制御構文（if 文，while 文）
- 整数・不動小数点・文字列
- 関数定義・関数呼び出し

4.3　ライブラリの移植と実装

TinyKonoha で使用するすべてのライブラリは静的にリンクする必要がある．コンパクト化の観点から使用する C 基本ライブラリ関数は，TinyKonoha のビルドに必要なもののみリンクする．**表 3** は，TinyKonoha をビルドするため必要であった libc ライブラリに相当する関数群である．

TOPPERS はリアルタイム OS であるため，動的メモリ管理機構はない．そのため malloc/free 関数には自作した

表 3　C 基本ライブラリに相当する関数群
Table 3　List of libc-compatible functions.

memcpy	newlib を使用
memset	newlib を使用
malloc	自作のライブラリを使用
free	自作のライブラリを使用

```
struct TinyObject {
  struct header {
    short flags;
    short type;
  };
struct TinyObject fields[3];
```

図 7　TinyObject の定義
Fig. 7　TinyObject.

表 4　オブジェクトのコンパクト化
Table 4　Reduction of kObject.

	Konoha	TinyKonoha
ヘッダ	3 ワード	1 ワード
フィールド	5 ワード	3 ワード

関数を使用している．これ以外の snprintf をはじめとする関数群はすべて memset/memcpy に置き換えて再実装している．

一方，ロボット制御に必要な関数は Konoha 側から FFI 機構で呼び出せるようにビルトインした．これにより，TinyKonoha のスクリプトからセンサの値を読むなどの基本動作が可能となった．

4.4　オブジェクトのコンパクト化と TinyGC

TinyKonoha のヒープ領域の削減のためには，オブジェクトの構造体のコンパクト化が必要となる．**図 7** に，Konoha におけるオブジェクト kObject をコンパクト化した TinyObject の構造体の定義を示す．

3.2 節で述べたとおり Konoha のオブジェクトのヘッダにはクラス情報，flags メンバ，オブジェクト固有のメタデータが含まれている．このうち，TinyKonoha では一部のフラグとクラスの情報のみが利用されるため，各オブジェクトのヘッダを 3 バイトから 1 バイトへとコンパクト化している．全体として，オブジェクトのサイズを 8 ワードから 4 ワードに削減した（**表 4**）．

また我々は TinyKonoha のためにシンプルな Mark-Sweep GC を実装した．オブジェクトのサイズは 16 バイトの固定長とし，オブジェクトのサイズごとに管理する手法を廃止した．メモリ削減の例として Konoha において Method 型オブジェクトをあげる．Konoha では Method オブジェクトの構造体サイズは 64 バイトであったが，コンパイル時に必要な引数や型の情報がフィールドの多くを占めていた．パーサ・コード生成機能を分離した TinyKonoha では Method 型オブジェクトにはコンパイル後のバイトコードのみ必要となるため，構造体のサイズを大幅に削減することができる．TinyGC は 4 kB のヒープを持ち，最大 256 個のオブジェクトが利用できる．

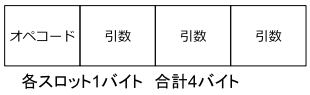

図 8　TinyVM のバイトコード
Fig. 8　Bytecode in TinyVM.

4.5　命令のコンパクト化と TinyVM

TinyVM は NXT 上で動作する TinyKonoha 向けのレジスタ型の VM である．

4.5.1　バイトコード命令長の変更

TinyKonoha のバイトコードの構造体レイアウトを**図 8** に示す．図 4 で示した MiniVM 用のバイトコードと比較してバイトコードのオペランドが削除されており，各オペランドのサイズも 4 バイトから 1 バイトに短縮され，バイトコードサイズが 32 バイトから 4 バイトへと削減されている．

MiniVM で用いられる Direct Threaded Code では命令の実行部分のアドレスを各バイトコード中に埋め込むことで命令ディスパッチの高速化を行う一方，バイトコードサイズは増加してしまっていた．そこで TinyVM では，Indirect Threaded Code を用いたディスパッチ方法を用いている．Indirect Threaded Code では 8 ビット整数値で表現されるオペコードを用いてジャンプ先アドレスのテーブルを引き，処理を行う．

また，Mini VM では，メソッド呼び出し命令と定数を扱う命令に関して，メソッドオブジェクトの参照や定数値をバイトコードへの埋め込み，実行の高速化を行っている．他方 TinyVM では 1 カ所にオブジェクトを集約し，8 ビットで表現されるインデックス値のみをバイトコードに埋め込む形をとることとした．ここで TinyKonoha ではヒープ領域として 4 kB が確保されているが，ヒープ中のオブジェクトすべてを 8 ビットで表現可能である．

4.5.2　TinyVM の命令セット

TinyVM ではスクリプトを実行するために最低限必要な 13 個の命令を基本命令セットに加えて，四則演算と比較に使用するバイトコードを新たに 12 個定義し，MiniVM の冗長な関数呼び出しの回数を削減している．

5.　評価

5.1　静的なメモリ利用量の評価

表 5 に，Konoha と TinyKonoha の実行コードサイズ，データ構造と各領域の違いを示す．

実行コードのサイズのみで 100 kB を超えていた Konoha

表 5　静的なメモリ利用量の評価
Table 5　Comparison of memory usage.

	Konoha	TinyKonoha
実行コードサイズ	131 kB	8 kB
オブジェクトの最小サイズ	32 バイト	16 バイト
オブジェクトのサイズ	可変（32, 64, . . .）	固定
ヒープ領域	16 MB	4 kB
バイトコード構造体サイズ	32 バイト	4 バイト

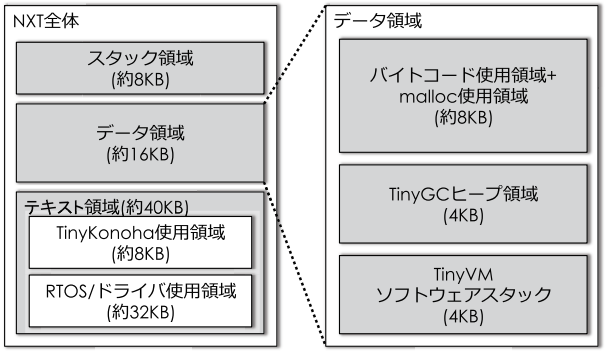

図 9　TinyKonoha のメモリレイアウト
Fig. 9　Memory layout of TinyKonoha.

のパーサを分離し，言語機能を Mindstorms NXT の制御のために必要最低限度に抑えた結果，実行コードサイズを大幅に削減している．また，TinyKonoha の実行時に特に多くの RAM を必要とするオブジェクトとバイトコードのレイアウトの見直しを行った．

図 9 に Mindstorms NXT に搭載された TinyKonoha のメモリレイアウトを示す．

TinyKonoha 全体の実行コードを格納するテキスト領域が最大の 40 kB を占めている．このうち TOPPERS にあたる部分が 32 kB を占めており，Mindstorms NXT 全体のほぼ半分の RAM を使用している．コンパイラ機能の分離により TinyKonoha の実行コードサイズを 8 kB にまで抑える結果となった．プログラム中の変数を格納するデータ領域のうち，Konoha スタック用の領域 4 kB と TinyGC が管理するオブジェクト用領域はそれぞれ 4 kB と設定されている．これらの領域長は固定されており，実行時の自動的な拡張は行われず，コールスタックや使用オブジェクト数が 4 kB に収まらなくなった時点でスクリプトは停止する．残るデータ領域は TinyKonoha のバイトコードやメモリアロケーション，その他の変数が使用しているが，これらの領域は記述したスクリプトの性質によって使用メモリ量が大きく変化するため，8 kB の上限のみを設けている．大きなスクリプトを実行させる場合はバイトコードが使用する領域が増加し，配列や文字列を多く使用するスクリプトの場合はオブジェクトのアロケーションとは別に配列の要素や文字列データを格納する領域が別途必要となるため，malloc により確保される領域が増加する．

表 6　12 行のライントレーススクリプトの使用メモリ
Table 6　Memory usage of a script of 12 lines.

行数	12 行
ソースコードサイズ	191 バイト（外部）
構文木の使用オブジェクト数	0 個
構文木の使用メモリ	0 kB
バイトコードの命令数	24 個
バイトコードサイズ	96 バイト
Konoha スタックの使用スロット	10 個

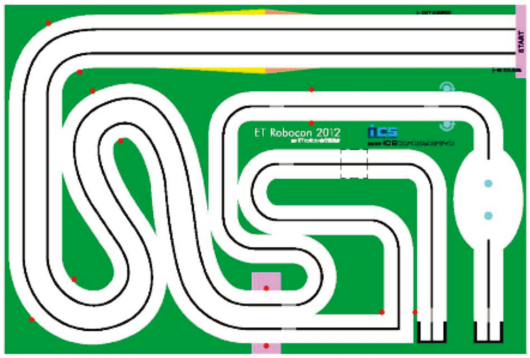

図 10　ET ロボコン 2012 公式コース
Fig. 10　Official course of ET Robot Contest 2012.

5.2　動的なコードの評価

TinyKonoha を実行する際のメモリ使用量が Konoha と比較してどの程度削減されているか，3.5 節のメモリ使用量の見積りと同様のスクリプトを用いて比較した結果を**表 6** に示す．

パーサ・バイトコード生成機能の分離により，ソースコードやコンパイル時に生成される構文木にメモリを必要としなくなっている．バイトコードの命令数には変化がないが，バイトコードサイズの削減によりメモリ使用量は減少している．また，TinyKonoha ではライントレーススクリプト中の NXT.getLightSensor 関数としきい値との比較が，関数コールから TinyVM の比較命令に変化しており，使用するスタックのスロット数の削減につながっている．

5.3　ET ロボコン 2012 公式コース

より複雑なスクリプトが動作することを確認するために，我々は ET ロボコン 2012 の公式コース [7] を走行するためのスクリプトを記述した（**図 10**）．スクリプトの詳細は以下のとおりである．本コースの走行には光センサを用いたライントレース走行に加えて，距離センサを用いたコース中の障害物の検知・回避行動などを記述する必要があるため，前述のライントレーススクリプトと比べて複雑な動作をスクリプトで記述することが要求される．4.5.1 項で示したとおり TinyKonoha のバイトコード長を 4 バイトにまで削減した結果，273 行のスクリプトのバイトコード領域を 2.5 kB にまで抑えている（**表 7**）．

表 7　ET ロボコン 2012 走行スクリプトの使用メモリ
Table 7　Memory usage of a script for ET Robot Contest 2012.

行数	273 行
バイトコードの命令数	626 個
バイトコードサイズ	2,504 バイト

6. 関連研究

TinyKonoha では実行コード削減のためにパーサ機能の外部への分離を行っているが，Lisp や FORTH [8] などパースが容易な文法を持つ言語の処理系をそのまま組み込み機器に搭載する試みも行われている．湯淺らは Mindstorms NXT の前身である Mindstorms RCX 上で，32 kB の RAM で動作する Lisp 処理系 XS [9] の開発を行っている．XS は TinyKonoha と同様省メモリ環境で動作する処理系を持つが，XS の場合は Lisp 処理系を新たに設計しており，Minxdstorms RCX 上の XS 処理系で Lisp のプログラムを直接評価する方式をとっている．

省メモリな環境で動作する軽量スクリプト言語として，Lua [10] や mruby [11] など，コンパクトな実装のスクリプト言語エンジンが開発されている．これらの言語は Mindstorms NXT と比較して規模の大きな組み込み機器上での実行が想定されている．また，センサネットワーク分野では，組み込みシステム用の最小化された設計の言語 [12] が開発されている．SwissQVM は，3 kB の SRAM メモリ上で動作するバイトコードインタプリタとして提案されている [13]．しかし，これには TinyOS 上で動作することが想定されており，十分な汎用性がない．

7. まとめ

本論文では ARM7CPU と 64 kB の RAM を搭載した Mindstorms NXT 上での静的型付け言語 Konoha の動作を目的として，Konoha 処理系のコンパクト化を行った TinyKonoha の設計と実装を行い，省メモリ環境におけるスクリプト言語処理系構築のための省メモリ化についてまとめた．実行コードとコンパイル時の構文木で使用されるメモリの削減のためにコンパイラ部分を外部に分離し，オブジェクトやバイトコードのデータ構造の簡略化などを行った結果，リアルタイム OS の機能や各種デバイスを利用したスクリプトを 64 kB の RAM 上で動作させることが可能となった．また，TinyKonoha で記述したスクリプトを用いて Mindstorms NXT のセンサとアクチュエータを使用したライントレーススクリプトに加えて，ET ロボコン 2012 公式コースを走行するスクリプトが動作し，Mindstorms NXT のデバイスを十分に活用するためのスクリプトを記述することが確認された．

参考文献

[1] Takada, H.: Introduction to the TOPPERS Project — Open Source RTOS for Embedded Systems, *Proc. 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC '03*, p.44, IEEE Computer Society (2003).

[2] Kuramitsu, K.: Konoha: Implementing a Static Scripting Language with Dynamic Behaviors, *Workshop on Self-Sustaining Systems, S3 '10*, pp.21–29, ACM (2010).

[3] Ueno, K., Ohori, A. and Otomo, T.: An Efficient Non-moving Garbage Collector for Functional Languages, *Proc. 16th ACM SIGPLAN International Conference on Functional Programming, ICFP '11*, pp.196–208 (2011).

[4] 養安元気，菅谷みどり，井出真広，倉光君郎：移動のない Incremental GC におけるリアルタイム性評価，技術報告 2，横浜国立大学 (2012).

[5] Ertl, M.A.: A portable Forth engine, *Proc. EuroFORTH'93 Conference* (1993).

[6] Shi, Y., Casey, K., Ertl, M.A. and Gregg, D.: Virtual machine showdown: Stack versus registers, *ACM Trans. Archit. Code Optim.*, Vol.4, No.4, pp.2:1–2:36 (2008).

[7] 一般社団法人組み込みシステム技術協会：ET ロボコン競技規約，入手先 ⟨http://www.etrobo.jp/2012/gaiyou/kiyaku.php⟩.

[8] Team, F.S.: *FORTH-79 Standard*, MVP-FORTH series, Forth Interest Group (1980).

[9] Yuasa, T.: XS: Lisp on Lego MindStorms, *International Lisp Conference* (2003).

[10] Ierusalimschy, R., de Figueiredo, L.H. and Celes, W.: The evolution of Lua, *Proc. 3rd ACM SIGPLAN conference on History of programming languages, HOPL III*, pp.2-1–2-26, ACM (2007).

[11] 特定非営利活動法人軽量 Ruby フォーラム：mruby，入手先 ⟨http://forum.mruby.org/⟩.

[12] Dunkels, A., Schmidt, O., Voigt, T. and Ali, M.: Protothreads: Simplifying event-driven programming of memory-constrained embedded systems, *Proc. 4th international conference on Embedded networked sensor systems, SenSys '06*, pp.29–42, ACM (2006).

[13] Müller, R., Alonso, G. and Kossmann, D.: A virtual machine for sensor networks, *Proc. 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, EuroSys '07*, pp.145–158, ACM (2007).

志田 駿介

1989 年生．2012 年横浜国立大学工学部卒業．言語処理系，組み込みシステム等の研究に従事．

井出 真広　（学生会員）

1987 年生．2012 年横浜国立大学工学府物理情報工学専攻修士課程修了．言語処理系の研究に従事．

倉光 君郎　（正会員）

1972 年生．愛知県出身．2000 年東京大学大学院理学系研究科情報科学専攻博士課程中途退学．同年東京大学大学院情報学環助手．2007 年より横浜国立大学工学部准教授．ユビキタスコンピューティング応用から，プログラミング言語 Konoha の研究開発に着手する．博士（理学），平成 20 年山下研究記念賞受賞．ACM，日本ソフトウェア科学会各会員．