対象のアプリケーションが使うデータ型に特化した JavaScript VM 開発フレームワーク

1215079　片岡 崇史　【プログラミング言語研究室】

A Framework for Development of JavaScript VMs Specialized for Datatypes That Target Application Uses

1215079　Takafumi Kataoka　【Programming Languages and Systems Lab.】

1　はじめに

JavaScript は生産性が高いプログラミング言語で，JavaScript のイベント駆動的なプログラミングスタイルは組込みシステム開発と親和性が高い．しかし，組込みシステムは一般的に計算資源が乏しいため，大きな仮想機械（以下，VM）を必要とする JavaScript を実行することは難しい．そこで本研究では，実行する JavaScript アプリケーションごとに特化した VM を作ることによって，VM のサイズを小さく，また高速にするというアプローチをとる．そのために，実行対象のアプリケーションの性質に合わせてカスタマイズした JavaScript VM を生成するためのフレームワーク eJSTK を開発した [1].

JavaScript の演算子は，オペランドに与えられた値のデータ型によって実行する処理が異なる．例えば，加算演算子「`+`」は，`1+2` のように両オペランドに数値が与えられた場合，数値の足し算として処理する．一方，`"a"+"b"`のように，文字列が与えられた場合は文字列連結が行われる．そのため VM では，実行をデータ型によって分岐する処理（以下，型ディスパッチ）が頻繁に行われる．

そこで，本研究で開発した eJSTK は，アプリケーションが使用するデータ型に注目した二つのカスタマイズを行う．一つは，VM 内部で扱うデータ型の表現をアプリケーションの特性に合わせることで型ディスパッチを高速化することである．もう一つは，対象のアプリケーションが必要としない演算子の機能を VM から省くことで，VM サイズを小さくすることである．例えば，あるアプリケーションでは，加算演算子を数値の足し算としてのみ使用する場合，文字列連結を行うコードを省くことができる．

しかし，アプリケーションが演算子のオペランドに与え得るデータ型を，プログラマが全て把握することは難しい．そこで，本研究ではこれを自動で特定するためのプロファイラも開発した．このプロファイラは，プログラマが作成したユニットテストを実行し，その中でアプリケーションが演算子のオペランドに与えたデータ型を記録する．これによって，比較的簡単にアプリケーションが必要とする演算子の機能を特定することができる．

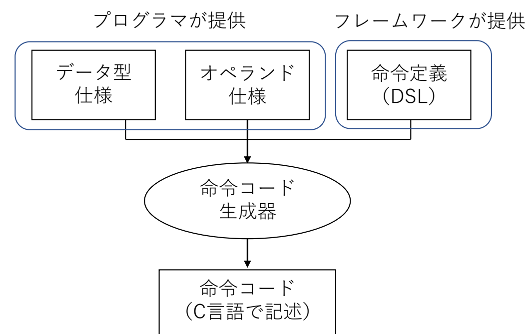

図 1　eJSTK のコード生成の流れ

2　データ型表現

以下では，eJSTK が生成した VM を eJSVM と呼ぶ．eJSVM は，専用の VM 命令の列を順に実行するバイトコードインタプリタの方式で動作する．

eJSVM は，VM 内部で扱う値のデータ型を二種類の型情報で区別する．一つは，データを指すポインタの下位 3 ビットで表現する「ポインタタグ」である．もう一つは，データのヘッダに含まれる「ヘッダタグ」である．ポインタタグは，データ本体にアクセスせずに取得できるため，ポインタタグを使えばヘッダタグを使うよりも高速に型を判別できる．しかし，ポインタタグは 3 ビットしかないため，区別できる型の数が限られている．そのため，頻繁に使用されるデータ型をポインタタグのみで判別できるようにすることが重要である．

3　eJSTK

図 1 に，開発した eJSTK のコード生成の流れを示す．データ型仕様は，VM 内部の各データ型に関して，ポインタタグとヘッダタグの値を指定する入力である．ここで，プログラマは頻繁に使用するデータ型のポインタタグの値を，他のデータ型と異なる値（以下，ユニークなポインタタグ）に設定する．オペランド仕様には，各 VM 命令についてオペランドに与えられ得るデータ

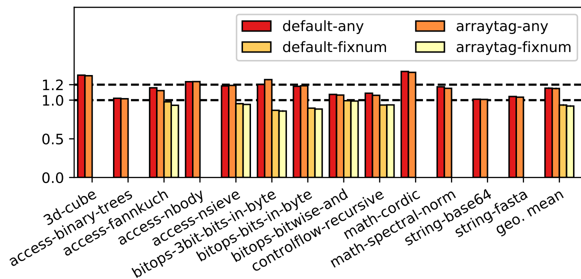

図 2 `handcrafted` を基準とした実行時間の比較

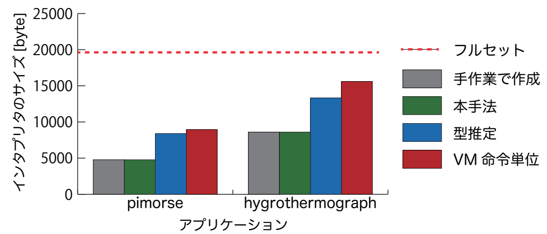

図 3 eJSTK が生成した eJSVM 全体のうちインタプリタ部分のサイズ

型を指定する入力である．例えば，加算演算子からコンパイルされる `add` 命令のオペランドを整数型のみに制限することができる．eJSTK は，この二つの入力と専用のドメイン特化言語で書いた命令定義から，特化した VM 命令のコードを生成する．

eJSTK が生成する型ディスパッチコードは，文献 [2] のアルゴリズムで生成する．生成される型ディスパッチコードは，データ型仕様でユニークなポインタタグが割り当てられたデータ型をポインタタグのみでデータ型を判別する．また，オペランド仕様によって制限されたオペランドのデータ型を処理するコードは省かれ，更に型ディスパッチコードからはこれに合わせて不要な分岐が削除される．

4 オペランド仕様を作るためのプロファイラ

VM 命令のオペランドに与えられ得るデータ型を特定するためのプロファイラを作成した．このプロファイラは，既存のユニットテストフレームワーク Jasmine を使って作成されたユニットテストを利用する．ユニットテストは普通，アプリケーションを網羅的に検査するように作られる．そのため，ユニットテストの実行トレースを取得できれば，実際にアプリケーションが VM 命令のオペランドに与えるデータ型を特定することができると考えたからである．

5 eJSTK の評価

eJSTK の性能を調べるため，二つのデータ型仕様と二つのオペランド仕様の組み合わせから eJSTK を使って 4 つの VM を生成し，これらと手動でチューニングした VM「`handcrafted`」を比較した．使用したデータ型仕様は，`handcrafted` と同じデータ型表現となる `default` と，`default` に比べて配列に特化した `arraytag` である．また，使用したオペランド仕様は，全ての VM 命令がどのデータ型も受理する `any` と，演算命令のオペランドに整数のみ受理する `fixnum` である．使用したベンチマークは，SunSpider ベンチマーク [1] に小さな変更を加えた 13 個のプログラムである．実験環境は，ARM プロセッサを搭載した Raspberry Pi 3 である．

図 2 に `handcrafted` を基準とした各 VM の実行時間を示す．ベンチマークプログラムのうち，名前の先頭に `bitops-`が付くものは，整数に対してビット演算を頻繁に行う．これらのプログラムは，オペランド仕様に `fixnum` を使用した VM の実行が `handcrafted` よりも高速であった．これは，VM 命令のオペランドのデータ型を制限したことによって，型ディスパッチがシンプルになったためである．

6 プロファイラの評価

オペランド仕様を作るために，本手法以外に型推定を用いる方法が考えられる．そこで，簡単な型推定プログラムを作成し，本手法と比較した．各方法でオペランド仕様を作成し，それを使って eJSTK で VM を生成する．eJSTK は基本的に VM のうちインタプリタの部分のみカスタマイズを行うため，本実験ではインタプリタのサイズのみを比較する．使用したプログラムは二つで，どちらも C++で書かれた組込みアプリケーションを JavaScript に移植したものである．

図 3 にインタプリタのサイズを示す.「フルセット」は，全てのデータ型のオペランドをサポートする VM である．本手法は型推定を用いた方法に比べて，インタプリタを大幅に小さくすることができた．

7 まとめ

本研究では，アプリケーションの使用するデータ型に注目して VM を作るためのフレームワークを開発した．入力が適切であれば，アプリケーションに特化して高速かつ小さな VM を生成することができる．

また，VM 命令のオペランドに与えられ得るデータ型を特定するためのプロファイラを作成した．本プロファイラで生成したオペランド仕様は，型推定を使ったものに比べて不要な機能を多く省くことができた．

参考文献

[1] T.Kataoka, T.Ugawa, and H.Iwasaki. A Framework for Constructing JavaScript Virtual Machines with Customized Datatype Representations. In *Proc. SAC* 2018, pp. 1238–1247, ACM(2018).

[2] T.Ugawa, H.Iwasaki, and T.Kataoka. eJSTK: Building JavaScript virtual machines with customized datatypes for embedded systems. *J.Virtual Lang. & Comp.* 2019 (to appear).

[1] https://webkit.org/perf/sunspider/sunspider.html