要 旨

仮想機械の型ディスパッチャ自動生成ツールの設計と実装

片岡 崇史

我々は，組込みシステム用の JavaScript 仮想機械を開発している．JavaScript は高機能なプログラミング言語であるために，JavaScript のプログラムを実行する仮想機械は大きくなりやすい．組込みシステムは一般的に CPU やメモリなどの計算資源が乏しく，このような環境で大きな JavaScript 仮想機械を動作させることは困難である．このような理由から，組込みシステム用の JavaScript 仮想機械は，そのサイズは小さく，実行が高速である必要がある．我々は，実行を高速にするために，仮想機械内部の型の表現方法に注目して，実行するプログラムごとに適した仮想機械を作ることを考えている．我々の開発している仮想機械では，型を二つの方法で表現している．一つはポインタの一部のビットで表される型情報（以下，ポインタタグ），もう一つはそのポインタが指すオブジェクトのヘッダに含まれる型情報（以下，ヘッダタグ）である．データの型を判別するとき，ポインタタグを使う方が高速に判別することができる．しかし，ポインタの中で型情報として使用できるビット数は少なく，ポインタタグのみでは全ての型を表現することができない．ポインタタグで表現されない型はヘッダタグで判別する．そのため，頻繁に使用する型をポインタタグで表現した方がプログラムの実行は高速になる．そこで，実行するプログラムごとに，そのプログラムにあった型の表現をする仮想機械を作る．型の表現方法の変更に影響を受ける処理として，型ディスパッチがある．型ディスパッチとは，与えられたデータの型によって実行する処理を変えることである．型ディスパッチを必要とする箇所は多く存在するため，型の表現方法を変えるたびにこれを全て書きなおすのは労力が大きい．

本研究では，型の表現方法の定義を与えることで，それにあった型ディスパッチを行うコードを含む仮想機械のソースコードを自動生成するツールを開発した．本ツールは，型の

表現方法の定義以外に，型ディスパッチを抽象化した仮想機械の命令定義を受けとる．仮想機械の命令の定義には本研究で開発した命令定義用ドメイン特化言語を用いる．本ツールは，仮想機械の命令ごとに，ソースコードをファイルに出力する．

本ツールが出力したコードを実際に仮想機械のソースコードに組込み，異なる型の表現から二つの仮想機械を作ることで本ツールを評価した．一つは文字列の扱いに特化した仮想機械，もう一つは配列の扱いに特化した仮想機械である．本ツールが出力したコードを使用した仮想機械は，75 のテストプログラムでテストし，正常に動作していることを確認した．また，本ツールによって異なる型の表現をする仮想機械を比較的簡単に作ることができた．さらに，ベンチマークプログラムとして，文字列を頻繁に扱うプログラムと，配列を頻繁に扱うプログラムを使用して，本ツールを用いて作成した仮想機械の性能を評価した．その結果，文字列を頻繁に扱うプログラムは，文字列の扱いに特化した型の表現をする仮想機械上で高速に実行された．同様に配列を頻繁に扱うプログラムも，配列の扱いに特化した型の表現をする仮想機械上で高速に実行された．これらのことから，本ツールの有用性が示された．

キーワード　仮想機械，JavaScript，プログラムの特化，組込みシステム

Abstract

Design and Implementation of a Type Dispatcher Generating Tool for Virtual Machines

Takafumi KATAOKA

We are developing a JavaScript virtual machine (VM) for embedded systems. JavaScript is a programming language that has rich facilities. Thus, JavaScript VMs tend to be large size. Typically, embedded systems are poor in computing resources such as CPU and memory. This makes it difficult to use JavaScript VMs on embedded systems. Therefore, JavaScript VM for embedded systems should be small and faster. To address this problem, we take an approach to make a dedicated VM for each JavaScript program that only has facilities of JavaScript that are used in the program in order to make the VM smaller and quicker. In this research, we focus on representation of types. Our VM represents types in two ways. Some types are represented by pointer tagging; we embed type information in spare bits of pointers to the objects. The others are represented by storing type information in the header of each object. Though type can be examined quicker with pointer tagging, the spare bits are too few to distinguish all types. The over all execution time of a program is short if we represent frequently used types in the program by pointer tagging. Therefore, we are planning to make dedicated VMs that use the appropriate type representation for each program. The type dispatcher depends on how types are represented. Type dispatcher selects an appropriate data processing routine based on the type of given data. However, it is time consuming to rewrite all type dispatch code every time we change representation

of types because JavaScript VM performs type dispatching in many places in its source code.

In this study, we developed a tool that automatically generates VM source code containing a type dispatcher corresponding to the given definition of type representation. Our tool receives the definition of the instructions of the VM where the type dispatching process is abstracted away. The definition of the instraction is described in a domain specific language (DSL) that we also developed. Our tool generates source code for each VM instruction into separate files.

We evaluated our tool by developing two VMs with different type representation using our tool. One of the type representation is specialized to frequent use of strings. The other is specialized to frequent use of arrays. These two VMs passed our 75 test cases. We also confirmed that our tool makes it easy to develop VMs that have different type representations. Furthermore, we mesured performance of the two VMs by using two benchmark programs; one is a program that uses strings frequently and the other is a program that uses arrays frequently. As a result we found that the VM specialized to use of strings was faster in the execution of the program that uses strings frequently, and the VM specialized to use of arrays was faster in the execution of the program that uses arrays frequently. This result indicates usefulness of our tool.

key words virtual machine, JavaScript, program specialization, embedded system