ネイティブコード安全性を保証するコード照合器付き ECC Scheme システム*

乗本 英介 伊藤 貴康 †
東北大学大学院情報科学研究科 ‡

1 はじめに

ネイティブコードによるモバイルコードの安全性を静的に検証する方式の一つに Efficient Code Certification（ECC）[2,3] がある．ECC は，ソースプログラムの構造に関する情報をネイティブコードに付加し，付加情報に基づいた検査を効率的に行うことにより，安全性を保証する手法である．

プログラミング言語 Scheme のサブセットに対して，検査のための付加情報付きのコードを生成し，そのコードの安全性を静的に保証する **ECC Scheme システム**の設計と実装を行った．

2 ECC Scheme システムの概要

ECC Scheme システムの構成を図 1 に示す．構造と意味がシンプルであり，付加情報の生成が容易である Scheme [1] のサブセットで書かれたソースプログラムを入力とし，検査のための付加情報である**注釈**が付加された注釈付きのインテル 32 ビットアーキテクチャ（IA-32）アセンブラ言語プログラムをネイティブコードとして出力する **ECC コンパイラ**と，生成された注釈付きコードの安全性を照合により検査する **ECC 照合器**から構成される．ECC 照合器により，コードが安全なものであると判断された場合は，ランタイムシステムとのリンクの後，実行が行われる．また，安全でないと判断された場合はエラーを報告する．ECC 照合器では次の 3 つの安全性を保証する：(1) 有効な命令列を含んだコード領域にのみ分岐する（**制御フローの安全性**）．(2) データ領域，明示的に確保したヒープ領域，有効なスタックフレームのみを操作する（**メモリの安全性**）．(3) スタックの状態がサブルーチン呼び出しにおいて保存される（**スタックの安全性**）．

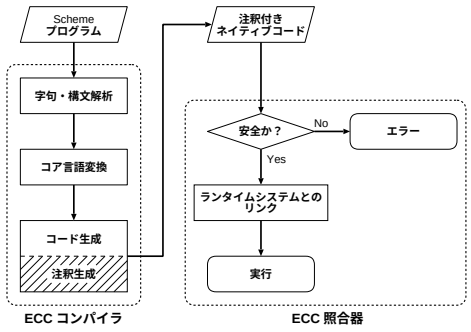

図 1: ECC Scheme システムの構成

表 1: ECC コンパイラが生成する主なブロック

ブロックの種類	目的
プログラムブロック	プログラム全体を表わす
メソッド初期化ブロック	`lambda` 式の評価
コールブロック	関数呼び出しのエントリ
関数本体ブロック	関数本体
呼び出し列	ユーザ定義関数呼び出し
If ブロック	`if` の評価
代入ブロック	`set!` の評価
評価ブロック	プリミティブの呼び出し
エラーブロック	ランタイムエラー

```
1 .begin program block
| 1 begin call block
| | 4 .begin assign block
| | | 4 .begin method initialization
| | | | 5 .begin call block
| | | | | 10 .begin method body
| | | | | | 10 .begin if block
| | | | | | |     ......
| | | | | | 29 .end if block status=InReg(eax)
| | | | | |                   type=Boolean
| | | | | 29 .end method body
| | | | 31 .end call block bound=[Unknown] args=1
| | | 39 .end method initialization type=Closure(1)
| | |                                status=InReg(ecx)
| | 42 .end assign block effect=(1,Closure(1))
| | 43 .begin call sequence
| | |             ......
| | 55 .end call sequence args=1
| 70 .end call block free=[box(Closure(1))] args=0
70 .end program block
```

図 2: 注釈の例

3 ECC コンパイラ

ECC コンパイラは `call/cc` 手続きの禁止，数として整数のみしか扱えないといった制限をもつ Scheme のサブセット言語に対するコンパイラである．ECC コンパイラは，サブセット言語プログラムを，まず，より単純な構文を持つ Scheme コア言語プログラムへと変換する．コア言語プログラムに対してコンパイルコードを生成すると同時に注釈を生成する．注釈は，関数呼び出しや式の評価などの機能単位毎に与えられ，生成されたコードの範囲を**ブロック**として明示化する．

注釈はコードの範囲，ブロックの種類，付加情報から構成される．ブロックの種類は基本的には Scheme の構文要素に対応する．表 1 に生成される主なブロックの種類を示した．付加情報はブロックの種類に応じて ECC 照合器で必要とされる情報であり，以下のものがある．

- `type`：ブロックの評価値の型．
- `status`：評価値の格納場所（レジスタなど）．
- `args`：仮引数または実引数の数．
- `bound`：ブロック内の自由変数の環境構造．
- `free`：ブロック内の束縛変数の環境構造．
- `effect`：代入における副作用．

注釈の例を図 2 に示す．この注釈は，

```
(define f (lambda (x) (if ...))) (f 5)
```

のようなプログラムに対して生成される注釈である．

---

* ECC Scheme system equipped with a code checker for the safety of mobile native codes

† Eisuke Norimoto, Takayasu Ito

‡ Department of Computer and Mathmatical Sciences, Graduate School of Information Sciences, Tohoku University

注釈はコードに付加されるため小さいことが望まれる．ECC コンパイラでは，生成されるオブジェクトコードにおいて，このような注釈が占める割合は 20〜30%程度である．

4 ECC 照合器

ECC 照合器では，まず注釈付きネイティブコードの注釈からコードのブロックを同定し，ブロックが正しく木構造をとっているかを検査する．また，ブロック毎に必要とされる付加情報があるかを検査し，省略された情報に関しては予め定められた値を補う．

次に，各ブロックに対して，ブロックに含まれるコードからサブブロックに含まれる命令列を取り除いたコードである**残分コード**を抽出する．残分コードにおいて，取り除かれた命令列は以下のことが仮定された一つの仮想的な命令として扱われる．

- 事前条件が満たされ，その実行が終了するならば，事後条件を満たす．
- 実行はすべて安全である．

事前条件，事後条件はブロックの種類と付加情報により決定する．例えば，図 2 の注釈において，$[10, 29]$ の If ブロックの事前条件は環境構造が 31 のコールブロックの付加情報に従うこと，事後条件はレジスタ `EAX` に `Boolean` 型の値が返されることである．また，実行が安全であるとは，スタックと環境構造が保存され，不正なメモリの操作を行わないことを意味する．

このような残分コードに対して，ECC 照合器はブロックの種類に応じて事前条件が満たされた状態の下で，以下の条件が満たされていることを確かめる．

1. すべての分岐が残分コード内，もしくは残分コードの次の命令へのものである．
2. サブブロックの事前条件を満たす．
3. ブロックからの脱出時には事後条件を満たす．
4. スタックと環境構造が保存されている．
5. メモリ操作がすべて安全である．

すべてのブロックの残分コードについてコードの照合により残分コードがこの条件を満たしているか否かを検査する．プログラム全体を表わす最外のブロックであるプログラムブロックがその事前条件を満たし，実行が最初の命令から行われるならば，プログラム全体が 2 節で述べた意味において安全であることがいえる．

ECC 照合器は，ブロック単位でコンパイラによって生成された付加情報に含まれるパラメータとブロック毎に予めメタ的に与えられたパラメータに従って，残分コードが条件 1〜5 を満たしていることを検査する．すなわち，ネイティブコードをブロックにより構造化することにより，付加情報による簡潔な検査を行い，各ブロックの残分コードの安全性からコード全体の安全性を検査するものである．そのため効率的な検査が可能となっている．しかし，ECC 照合器で行っているのは，実行ブロックに基づく安全性を検査するものであり，データや計算の意味を考慮した安全性の検証ではないことに注意されたい．

表 2: ECC Scheme システムの性能

プログラム	IA-32 命令数	コンパイラ	検査時間	SCM
fib	313	0.62s	0.003s	11.04s
tak	469	0.06s	0.003s	1.48s
hanoi	503	0.02s	0.006s	0.53s
queen	1269	0.09s	0.012s	0.58s

実験環境 CPU:Pentium 4 1.8GHz OS:FreeBSD 4.6.2

5 実験評価

ECC Scheme システムは IA-32 に従うプロセッサを搭載した PC で動作する FreeBSD 上で実現されている．本システムの性能をいくつかの簡単なベンチマークプログラムを用いて測定したところ，表 2 のような結果が得られた．実験環境は CPU:Pentium 4 1.8GHz，OS:FreeBSD 4.6.2 である．「IA-32 命令数」,「コンパイラ」はそれぞれ生成されたネイティブコードの命令数と実行時間を表わす．比較対象として用いた SCM は Scheme の処理系の一つであり，インタプリタである．ベンチマークプログラムは以下のものを用いた．

- fib：フィボナッチ数列
- tak：竹内関数
- hanoi：ハノイの塔
- queen：n-queen 問題

まず，この結果から ECC 照合器において検査にかかる時間は命令数にほぼ比例することがわかる．これは ECC 照合器による安全性の検査がコードの照合による単純なものであり，複雑なフロー解析などを必要としないからである．

また，ECC Scheme システムにより生成されるコードがインタプリタである SCM より，6〜25 倍程度高速であることがわかる．これは，インタプリタにより動的に安全性の検査する方式に比べ，本システムのようなネイティブコードに対して静的な検査を行う方式の実行速度上の優位性を示す結果といえる．

6 おわりに

本稿では，実現を行った Scheme で記述されたプログラムの実行コードの安全性を，ECC のアイディアに基づいて静的に検査する ECC Scheme システムの概要と実験結果について報告し，効率的な照合検査を行うことができることを述べた．

Scheme 以外の他の言語への本稿で述べた手法の適用，現在は高性能 PC 上で実現されているシステムの携帯端末での利用，Web ブラウザへの本システムの組み込みなどは今後の課題ある．

参考文献

1) *IEEE Standard 1178–1990. IEEE Standard for the Scheme*. IEEE, New York, 1991.

2) Dexter Kozen. Efficient Code Certification. Technical Report 98-1661, Computer Science Department, Cornell University, January 1998.

3) Dexter Kozen. Language-based Security. In M. Kutylowski, L. Pacholski, and T. Wierzbicki, editors, *Proc. Conf. Mathematical Foundations of Computer Science (MFCS'99)*, Vol. 1672 of *Lecture Notes in Computer Science*, pp. 284–298. Springer-Verlag, September 1999.