特別研究報告書

スマートコントラクトのガス消費量の Reource Aware ML を用いた静的解析

指導教員：末永　幸平　准教授

京都大学工学部情報学科

小野　雄登

2021 年 2 月 2 日

スマートコントラクトのガス消費量の Reource Aware ML を用いた静的解析

小野　雄登

内容梗概

2009 年にビットコインを用いた取引がオープンソフトウェアで始まって以来，現在に至るまでにブロックチェーンを技術基盤とする様々な仮想通貨が開発されている．スマートコントラクトは，仮想通貨の取引における契約の締結や履行を自動化する仕組みであり，ブロックチェーン上で動作するプログラムとして実装される．

スマートコントラクトにはガスの概念が存在し，ガスはコントラクトの実行にかかる手数料を表している．コントラクトが実行される際に，コントラクトの各命令の評価毎に命令の内容に比例した量のガスが消費され，消費量の合計が許容ガス消費量を超えると，その命令が直ちに停止され，命令の実行による値の変更が取り消される．プログラムとして非効率なコントラクトが実行されると，想定される量以上のガスが消費されてしまうので，コントラクトのガス消費量を静的に解析することは，ユーザーが必要以上に手数料を支払わないために必要な技術であると考えられる．

本研究では，スマートコントラクトのガス消費量の静的な解析を行うプログラムを実装する．具体的には，スマートコントラクトを実装しているブロックチェーンプロトコルである Tezos において，スタックベースのプログラミング言語 Michelson で書かれたコントラクトのガス消費量を，プログラミング言語型のツールである Resource Aware ML(RAML) を用いて静的に解析する．RAML は，OCaml で用いられる文法を備えた関数型プログラミング言語で，入力として与えられたプログラムのリソース消費量の上界を，指定されたメトリックに従って自動的に，かつ静的に解析して，その結果を出力するツールである．

実装の方針は，

1. Michelson の挙動を再現するためのライブラリを RAML で設計する．
2. 設計したライブラリを用いて，Michelson で書かれたコントラクトをエンコードする．
3. エンコードしたコントラクトを解析し，ガス消費量を見積もる．

という流れである．以下，各過程について説明する．

1. において，Michelson はスタック構造をもち，コントラクトに用いられる命令は，初期スタックを受け取ってスタックの内容を変更して返す関数として実装されている．この構造を RAML で設計するにあたって，スタックの要素をヴァリアント型 t として定義し，命令を (t list -> t list) 型をもつ関数として定義した．

2. において，Michelson のコントラクトは，初期スタックに入る値の型宣言と，初期スタックに対して順に適用される一連の命令によって構成されている．RAML においてこのコントラクトを，1. で設計したライブラリを用いて，初期スタックを表すリストに対して命令を順に関数適用するプログラムとして実装した．

3. において，2. で実装したプログラムを，RAML の steps メトリックを用いて評価ステップ数に関する解析を行った結果，基本的なスタックの操作に関する命令や，簡単な算術演算や条件分岐の命令のみを含むコントラクトについては解析が正しく行われたが，ループ命令や，リストや集合に対する再帰を行う命令を含むコントラクトについては解析が失敗するものも存在した．続いて，ガス消費量の見積もりについては，RAML の tick メトリックを用いた．tick メトリックは，リソース消費の値や発生するタイミングを，ユーザーが関数として定義することができるメトリックである．RAML で実装した各命令について，その命令のガス消費量に相当する値の tick 関数を定義し，tick メトリックを用いた解析を行い，コントラクトのガス消費量を見積もれるかどうかを検証した．結果として，コントラクトの実行において発生するガス消費のうち，プログラムの解釈実行を行う際に発生する interpreter cost について概ね正しく消費量を見積もることができた．

本研究においては，Michelson に実装されているコントラクトの命令のうち，主要なものについて RAML で実装した．残りの命令の実装については，今後の課題とする．また，ガス消費量の見積もりについては interpreter cost についてのみ取り組んだが，他の過程において発生するガス消費量の見積もり，ひいてはコントラクトの実行において発生するガス消費量全体の見積もりについても検討していきたい．

Static Analysis for Gas Consumption of Smart Contracts Using Resource Aware ML

Yuto Ono

Abstract

スマートコントラクトのガス消費量の Reource Aware ML を用いた静的解析

目次

1 序論

2009年にビットコインを用いた取引がオープンソフトウェアで始まって以来，現在に至るまでにブロックチェーンを技術基盤とする様々な仮想通貨が開発されている．取引の記録をブロックとしてネットワーク上に記憶するという性質上，ブロックチェーンはデータ改竄に対する優れた耐性を持ち，仮想通貨の取引を支えるコア技術となっている．

ブロックチェーン上で用いられる技術としてスマートコントラクトがある．スマートコントラクトは，仮想通貨の取引における契約の締結や履行を自動化する仕組みであり，ブロックチェーン上で動作するプログラムとして扱われる．第3者を介さずに，また相手の信頼を必要とせずに取引を行うことができ，決済期間の短縮や手数料の削減などの効果が期待できる．スマートコントラクトにはガスの概念が存在し，ガスはコントラクトの実行にかかる手数料を表している．コントラクトが実行される際に，コントラクトの各命令の評価毎に命令の内容に比例した量のガスが消費され，消費量の合計が許容ガス消費量を超えると，その命令が直ちに停止され，命令の実行による値の変更が取り消される．ガスの消費量の計算は複雑で，前もってガスの消費量を正確に見積もることは難しいとされているが，プログラムとして非効率なコントラクトが実行されると，想定される量以上のガスが消費されてしまうので，コントラクトのガス消費量を静的に解析することは，ユーザーが必要以上に手数料を支払わないために必要な技術であると考えられる．

本研究では，仮想通貨Tezosのスマートコントラクトのガス消費量の静的な解析を行うプログラムを実装した．Tezosはスマートコントラクトを用いたブロックチェーンを技術基盤とする仮想通貨の1つで，コントラクトはスタックベースのプログラミング言語Michelsonで書かれている．コントラクトのガス消費量はMichelsonプログラムの実行内容によって計算されるので，このプログラムに対して解析を行うことでガス消費量の見積もりを試みた．

解析の方法として，プログラミング言語型のツールであるResource Aware ML(RAML)を用いる．RAMLは，OCamlで用いられる文法を備えた関数型プログラミング言語で，入力として与えられたプログラムのリソース消費量の上界を，指定されたメトリックに従って自動的に，かつ静的に解析して，その結果を出力するツールである．具体的な方法としては，Michelsonで実装されて

いる型や命令などのライブラリを RAML で実装し，Michelson のコントラクトを RAML 上でエンコードし，それを解析してガス消費量の見積もる．

本報告書は以下のように構成されている．第 2 節では，本研究の背景知識として，Tezos と Michelson，スマートコントラクトにおけるガス消費の仕組み，そして解析に用いるツールである RAML についてそれぞれ説明する．第 3 節では，Mihcelson プログラムの RAML での実装について説明する．第 4 節では，第 3 章で実装した RAML プログラムを用いて行ったガス消費量の解析について，結果と考察を記述する．第 5 節では，実装したプログラムについていくつかの改善点を示す．最後に第 6 節で本研究についての結論を述べる．

2　背景知識

本節では，本研究に関連する背景知識について述べる．第 3.1 節では Tezos と Michelson について，第 3.2 節ではスマートコントラクトにおけるガス消費の仕組みについて，第 3.3 節では RAML についてそれぞれ述べる．

2.1　Tezos と Michelson について

Tezos はスマートコントラクトを用いたブロックチェーンを技術基盤とする仮想通貨の 1 つである．Bitcoin や Ethereum といった仮想通貨が先立って流通されるようになった中，Tezos はそれらのブロックチェーンの弱点を解消することを目的として開発された．Tezos の特徴として"Proof-of-Stake"(PoS) と呼ばれるコンセンサスアルゴリズムを採用していることが挙げられる．従来のブロックチェーンで採用されている"Proof-of-Work"(PoW) が，コンピュータの計算能力が高いユーザーに対してブロック生成の権利を与えているのに対して，PoS では通貨の保有量が多いユーザーに対してブロック生成の権利が与えられる．Tezos の採用している PoS は"Liquid-Proof-of-Stake"(LPoS) といい，ブロック生成の権利を他のユーザーに委任することができる．これにより，多くのユーザーがブロック生成に参加することができ，プロトコルの分散性を高めるという目的に寄与している．

Mihcelson は，Tezos のスマートコントラクトを記述するために用いられるプログラミング言語である．この言語はスタックベースで，高レベルのデータ型とプリミティブ，および厳密な静的型チェックを備えている．

Michelson のプログラムは，プログラムに対して与えられるパラメータと，ブロックチェーン上に保存されているストレージのペアを受け取り，このプログラムの終了後に実行される操作のリストと，プログラムの実行中に更新されブロックチェーン上に保存されるストレージのペアを返す純粋な関数である．プログラムの本体は，順番に実行される一連の命令である．各命令は，スタックを入力として受け取り，そのスタックの内容を書き換えて出力する関数である．プログラムの初期スタックは，引数として与えられたパラメータとストレージのペアが一番上に積まれた状態のもので，その初期スタックに対して順番に命令が適用され，最後に操作のリストとストレージのペアが一番上に積まれた状態のスタックが残り，それが出力される．

Michelson プログラムの例として，簡単な演算を行うプログラムである example1 のコードを Code 1 に示す．これは，整数のペアをパラメータとして受け取り，2 つの整数の和をストレージに書き込むプログラムである．

```
1  parameter (pair int int);
2  storage int;
3  code  /* ((para1, para2), st) */
4    { CAR ;   /* (para1, para2) */
5      DUP ;   /* (para1, para2) : (para1, para2) */
6      CAR ;   /* para1 : (para1, para2) */
7      DIP { CDR } ;   /* para1 : para2 */
8      ADD ;   /* st' */
9      NIL operation ;   /* [] : st' */
10     PAIR    /* ([], st') */
11   }
```

Code 1: example1.tz

以下，プログラムの内容を説明する．

Michelson プログラムのコードは，プログラムに対して与えられる引数である parameter と，ブロックチェーン上に保存されているストレージの値である storage の型宣言から始まる．1 行目は parameter の型が (pair int int) であること，2 行目は storage の型が int であることを宣言している．3 行目以降はプログラムの本体である code である．code には一連の命令が記述されていて，この命令が初期スタックに対して順に実行されていく．以下，code の内容について行番号ごとに説明する．なお，スタックの状態を記述する際，スタックの要素を : で区切って表す．

- プログラム開始時のスタックは，parameter と storage の値のペアが空のス

タックに積まれた状態で始まる．parameter の値を (para1, para2)，storage の値を st と表すとすると，初期スタックは ((para1, para2), st) である．

- 4 行目の CAR は，スタックのトップの要素がペアだった場合，その要素を取り出して，ペアの第 1 要素をスタックに積む命令である．4 行目時点でのスタックは (para1, para2) である．
- 5 行目の DUP は，スタックのトップの要素を複製してスタックに積む命令である．5 行目時点でのスタックは (para1, para2) : (para1, para2) である．
- 6 行目は 4 行目と同じく CAR を実行する．6 行目時点でのスタックは para1 : (para1, para2) である．
- 7 行目の DIP は引数として命令列 body を受け取る命令で，スタックのトップの要素を保持した状態で，その要素を取り出した状態のスタックに対して body を実行する命令である．つまり，スタックのトップの para1 を保持した状態で，スタック (para1, para2) に対して CDR を実行する．CDR は，スタックのトップの要素がペアの場合，その要素を取り出して，ペアの第 2 要素をスタックに積む命令である．よって，7 行目時点でのスタックは para1 : para2 である．
- 8 行目の ADD は，スタックのトップの要素 x と 2 番目の要素 y の型が，それら 2 つの演算が定義されているような型である場合，x と y を取り出し，x+y の値をスタックに積む命令である．例えば，x と y がともに int 型ならば，x+y は int 型である．para1 + para2 の値を st′ と表すとすると，8 行目時点でのスタックは st′ である．
- 9 行目の NIL は引数として型 a を受け取る命令で，リストの型が a であるような空のリスト [ ] をスタックに積む命令である．9 行目時点でのスタックは [ ] : st′ である．
- 10 行目の PAIR はスタックのトップの要素と 2 番目の要素を取り出し，それらのペアをスタックに積む命令である．この命令後，スタックは ([ ], st′) となり，プログラムが終了する．

プログラムの終了時のスタックは，このプログラム終了後に続いて行われる操作のリスト operation list と，実行中に更新されブロックチェーンに保存されるストレージの値 storage′ のペアのみが積まれている状態でなければならない．このとき，プログラム実行前のストレージの値 storage と，プログラム実行後のストレージの値 storage′ の型が一致している必要がある．

Michelson には静的型チェックが備えられている．それぞれの命令には，命令の実行前と実行後のスタックの状態を型で表した型規則が存在する．例えば，PAIR についての型規則は以下のように表される．

$$\mathrm{a}' : \mathrm{b}' : \mathrm{S}' \to \mathrm{pair}\ \mathrm{a}'\ \mathrm{b}' : \mathrm{S}'$$

これは，実行前のスタックのトップの要素の型が a′，2 番目の要素の型が b′ のとき，実行後のスタックのトップの要素が pair a′ b′ となることを示している．また，ADD についての型規則は以下のように表される．

$$\mathrm{int} : \mathrm{int} : \mathrm{S}' \to \mathrm{int} : \mathrm{S}'$$

これは，実行前のスタックのトップの要素と 2 番目の要素がともに int である必要があり，その場合，実行後のスタックのトップの要素が int となることを示している．

命令を実行する前にスタックの状態が型規則に則った形でない場合，命令は失敗する．これを防ぐために，プログラム実行前に静的な型チェックが行われる．

2.2 コントラクトのガス消費の仕組み

2.3 Resource Aware ML について

Resource Aware ML(RAML) は，一階の関数プログラムの多項式リソース消費量の上界を，静的かつ自動的に計算する関数型プログラミング言語である．プログラムの文法は OCaml のものを採用しており，入力として OCaml の文法で書かれたプログラムを与えると，その多項式リソース境界を出力するツールとして扱うことができる．

図 1 に RAML の文法を示す．e は RAML プログラム上の式を表していて，Unit 値 ()，Boolean 値 True，False，整数値 n，変数 x，変数 x_1 と x_2 の演算，関数適用，let 式を用いた局所関数を伴う式，if による条件分岐式，変数 x_1 と x_2 のペア，ペアに対する match 式，空リスト nil，リストの結合を表す cons(x_h, x_t)，リストに対する match 式がある．演算子 $binop$ には，整数値に対する演算である+，-，*，mod，div，Boolean 値に対する演算である and，or がある．A と F は，RAML プログラム上での simple type を表している．A はデータ型で，Unit 型の unit，Boolean 型の bool，整数型の int，simple type の値のリスト，simple type の値のペアがある．F は関数型で，simple type の値を受け取って

$$
\begin{aligned}
e &::= () \mid \mathrm{True} \mid \mathrm{False} \mid n \mid x \\
&\quad \mid x_1\ \mathit{binop}\ x_2 \mid f(x_1, ..., x_n) \\
&\quad \mid \mathrm{let}\ x\ =\ e_1\ \mathrm{in}\ e_2 \\
&\quad \mid \mathrm{if}\ x\ \mathrm{then}\ e_t\ \mathrm{else}\ e_f \\
&\quad \mid (x_1, x_2) \mid \mathrm{match}\ x\ \mathrm{with}\ (x_1, x_2) \to e \\
&\quad \mid \mathrm{nil} \mid \mathrm{cons}(x_h, x_t) \\
&\quad \mid \mathrm{match}\ x\ \mathrm{with} \mid \mathrm{nil} \to e_1 \mid \mathrm{cons}(x_h, x_t) \to e_2 \\
\mathit{binop} &::= + \mid - \mid * \mid \mathrm{mod} \mid \mathrm{div} \mid \mathrm{and} \mid \mathrm{or}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{A} &::= \mathrm{unit} \mid \mathrm{bool} \mid \mathrm{int} \mid \mathrm{L(A)} \mid (\mathrm{A}, \mathrm{A}) \\
\mathrm{F} &::= (\mathrm{A}, ..., \mathrm{A}) \to \mathrm{A}
\end{aligned}
$$

図 1: RAML の文法

simple type の値を返す関数型を表している．また，RAML 上の well-typed な式を，この simple type が割り当てられた式と定義している．

RAML プログラムは，関数宣言のリストと main 式からなる．関数宣言は，関数の型宣言または関数の定義であり，それぞれの関数定義に対して 1 つずつ型宣言を行うことができるが，プログラム内で型宣言が行われていない場合，プログラム実行時に型推論が行われる．main 式はリソース消費量の分析の対象となる式で，プログラムの最後に記述する．

RAML のリソース消費量の分析は，入力されたプログラムの，big-step operational semantics による各評価ステップに対して一定のコストを割り当てるメトリックによって，リソース消費量の計算を行う．メトリックは以下の 4 つが存在する．

- heap メトリックは，実行時に割り当てられたヒープセルの数を計算する．
- steps メトリックは，実行時の評価ステップ数を計算する．
- tick メトリックは，ユーザーが定義した tick 関数による tick 値を計算する．ユーザーは関数の定義中に Raml.tick(1.0) のような関数 (tick 関数) を定義することができる．tick 関数が呼び出される度に，引数の float 値に等しい

リソース消費 (tick 値) が発生する．

- flips メトリックは，フリップ関数によるフリップ数を計算する．本論文では扱わないため，詳細な説明は省略する．

ユーザーは分析を行う際にメトリックを指定することで，自分の注目するリソースの消費量を分析の出力として得ることができる．

RAML プログラムの例として，リストに対するクイックソートを行うプログラムである quicksort のコードを Code 2 に示す．

```
1  let rec append l1 l2 =
2    match l1 with
3      | [] -> l2
4      | x::xs -> x::(append xs l2)
5
6  let rec partition f l =
7    match l with
8      | [] -> ([],[])
9      | x::xs ->
10       let (cs,bs) = partition f xs in
11       Raml.tick(1.0);
12       if f x then (cs,x::bs) else (x::cs,bs)
13
14 let rec quicksort gt = function
15   | [] -> []
16   | x::xs ->
17       let ys, zs = partition (gt x) xs in
18       append (quicksort gt ys) (x :: (quicksort gt zs))
19
20 let _ = quicksort (fun a b -> a <= b)  [9;8;7;6;5;4;3;2;1]
```

Code 2: quicksort.raml

見てわかる通り，プログラムのコードの見た目は OCaml に近いが，20 行目の let _ = ... の部分は OCaml には見られない表現である．この式が main 式で，リソース消費量の分析の対象となる．このプログラムは，append，partition，quicksort の 3 つの関数を定義し，main 式は quicksort の関数適用が記述されている．

1-4 行目の append 関数は，2 つのリスト l1,l2 を引数として受け取り，それらを結合したリストを返す関数である．6-12 行目の partition 関数は，リスト l と，l の要素の型の値を受け取って bool 型の値を返す関数 f を引数として受け取り，l の要素を f に関数適用したときの返り値によって 2 つのリストに分割する関数である．11 行目に tick 関数である Raml.tick(1.0) があり，結果として l の要素の数だけ 1.0 の tick 値が発生する．14-18 行目の quicksort 関数は，リスト l と，

l の要素の型の値を 2 つ受け取って bool 型の値を返す関数 gt を引数として受け取り，l に対してクイックソートを実行する関数である．quicksort の定義中に append と partition が用いられている．

3 RAML での Michelson プログラムの実装

4 ガス消費量の解析の結果と考察

5 改善点

6 結論

謝辞

参考文献

[1] Caplener, H. D. and Janku, J. A.: Improved Modeling of Computer Hardware Systems, *Computer Design*, Vol. 12, pp. 59–64 (1973).

[2] Beizer, B.: Towards a New Theory of Sequential Switching Networks, *IEEE Trans. Computers*, Vol. C–19, pp. 936–956 (1970).

[3] 村上伸一: 微分方程式の解曲線の表示, 情報処理, Vol. 14, pp. 231–238 (1970).

[4] 平井有三, 福島邦彦: 両眼視差抽出機構の神経回路網モデル, 信学論 (D), Vol. 56–D, pp. 465–472 (1973).

[5] Baraff, D.: Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation, *SIGGRAPH '90 Proceedings* (Beach, R. J.(ed.)), Dallas, Texas, ACM, Addison-Wesley, pp. 19–28 (1990).

[6] 對馬雄次ほか: ボリュームレンダリング専用並列計算機のアーキテクチャ, 並列処理シンポジウム JSPP'94, pp. 89–96 (1994).

[7] Barnett, S. and Storey, C.: *Matrix Methods in Stablity Theory*, Nelson, London (1970).

[8] J. E. ホップクロフト, J. D. ウルマン（木村, 野崎訳）: 言語理論とオートマトン, サイエンス社, chapter 6 (1972).

[9] 寺沢寛一: 自然科学者のための数学概論, 岩波書店, pp. 325–328 (1955).