社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

再帰を含むプログラムに対するパラメトリック実行時間解析手法とツールの試作

椛島　和宏†　　中田　明夫‡

†‡広島市立大学　大学院情報科学研究科　システム工学専攻

〒731-3194 広島県広島市安佐南区大塚東 3-4-1

E-mail:　†kabashi@sos.info.hiroshima-cu.ac.jp,　‡nakata@hiroshima-cu.ac.jp

あらまし　実時間性を持つソフトウェア部品に関しては，実行時間がプロセッサなどの実行環境に依存するため，異なる環境への再利用は一般に困難である．パラメトリック実行時間解析手法を用いて，プログラムの実行時間を個々の処理の実行時間と繰り返し回数とのパラメータを用いた式の形で求めることにより，異なる実行環境においても実時間制約を満たすためのパラメータの最適化を，数理計画法を用いて容易に行うことができる．しかし，従来手法は有限オートマトンに基づいており，再帰呼び出しがあるプログラムは扱えなかった．本研究では，一般化 Parikh の定理に基づき，再帰を含むプログラムを扱うことを可能にしたパラメトリック実行時間解析手法を提案する．

キーワード　組込みソフトウェア，再利用，実時間制約，最悪実行時間解析，パラメータ最適化

Parametric Execution Time Analysis for Recursive Programs and its Experimental Evaluation

Kazuhiro Kabashima †　　Akio Nakata ‡

†‡ Department of Systems Engineering, Graduate School of Information Sciences, Hiroshima City University

E-mail:　†kabashi@sos.info.hiroshima-cu.ac.jp　‡nakata@hiroshima-cu.ac.jp

Abstract　It is generally difficult to reuse real-time software components to different execution platforms, since its execution time depends on the execution platforms such as processors. Parametric execution time analysis makes such reuse easier by deriving the execution time of a program as a function of its parameters such as the execution time of each atomic action and loop execution counts, and optimizing the parameters to fit the different platform using mathematical programming. However, the existing parametric execution time analysis is based on finite automata, and thus is unable to apply to programs containing recursion in general. In this paper, we propose a new parametric execution time analysis method that can handle programs containing arbitrary recursions based on the generalized Parikh theorem.

Keyword　embedded software, software reuse, real-time constraint, worst case execution time (WCET)analysis, parameter optimization

1. まえがき

近年，組込みソフトウェアの高機能化・高性能化が進んでいる一方で，市場競争により，限られた開発期間で，実時間制約の充足と低コスト化，低消費電力化を達成するソフトウェア開発手法が求められている．

そのような大規模化する実時間ソフトウェアの開発期間短縮のためには，通常のソフトウェアと同様にソフトウェア部品の再利用が有効であると考えられる．しかし，特に実時間性を持つソフトウェア部品に関しては，実行時間がプロセッサなどの実行環境に依存するため，異なる環境への再利用は一般に困難である[1].

例えば，実時間動画圧縮ソフトウェアや暗号処理ソフトウェアなどにおいては，圧縮率，解像度，暗号強度など，それを変更することによってソフトウェアの機能を損なわず，処理の品質のみに影響する設計パラメータが存在している．それらのパラメータの多くはプログラムの特定箇所の繰り返し回数と関連している．それらのパラメータ，および CPU の動作周波数の最小限の調整により，新しい動作環境でプログラムが実時間制約を満たし，かつ低消費電力で実行できるように

する最適設計支援手法が望まれる.

ソフトウェアの実時間制約の保証のために，従来，プログラムの最悪実行時間(Worst Case Execution Time: WCET）解析手法[2][3]が研究されてきた．これらの既存の WCET 解析手法を実時間ソフトウェアの再利用に用いる場合，次のような問題が生じる：

・プロセッサ､バス､メモリなどのハードウェアや (システムコールを利用しているソフトウェアの場合)OS などの実行環境を指定しなければ正確な実行時間の見積もりが困難である.
・解析の結果，実時間制約を満たさないと判明した場合，その制約を満たすように修正を行う一般的な指針がない
・再利用対象のソフトウェアを最適に実行できる動作環境の仕様決定に用いたい場合には，実行環境を変えながら試行錯誤で解析するしかない

これらの問題を解決するために，パラメトリックプログラム実行時間解析手法[4]が提案された．文献[4]は，単一 CPU 単一タスクの実行環境で動作する実時間ソフトウェアを対象とし，

・実時間プログラムのソースコード
・実行環境の情報
・プログラムの実行時間に関する実時間制約

が与えられたときに，一般にプログラム及び実行環境のパラメータに関する多項式の形で，実時間制約を満たして解析対象のプログラムの最悪実行時間の見積もりを自動導出するというものである．ここで，プログラムのパラメータとは，プログラムの指定した個所の繰り返し回数であるとし，実行環境のパラメータとしては，CPU の動作周波数および各命令後の実行に要する主要クロック数の上限および下限であるとする．従来の WCET 解析手法などでは，実行環境パラメータを具体的に与える必要があり，それらの最適値は試行錯誤で探す必要があった．しかしこの手法では，実時間制約を満たすための最適なパラメータの条件を選択することが可能となる.

文献[4]の手法では，プログラムの制御構造を有限オートマトンの形で抽出し，正規表現に変換することによりプログラムの繰り返し構造を導出し，プログラムの実行時間をループ回数の関数として導出している．しかし，再帰呼び出しを含むプログラムのクラスは有限オートマトンで表現可能なプログラムのクラスより大きいことが知られており[7]，一般に再帰呼び出しがあるプログラムは扱えないという問題があった.一方，文献[5]では，正規表現の代数的モデルである Kleene 代数において連接演算（積）を可換とした代数（可換 Kleene 代数）を用いて，任意の文脈自由言語に対してその語の終端記号の順序を入れ替えても等価とみなした語集合が正規表現で表現可能であることが示されており（一般化 Parikh の定理），その正規表現を求めるアルゴリズムが提案されている.任意の再帰を含むプログラムは文脈自由文法で表現可能であり，プログラムの実行時間は（キャッシュやパイプライン実行などによる微妙な影響を無視すれば）個々の単位動作の実行時間の合計のみに依存し，その実行順には依存しないため，文献[5]の結果を援用することにより，任意の再帰を含むプログラムの繰り返し構造を表現する正規表現を導出することができる.

本研究では，可換 Kleene 代数における一般化 Parikh の定理[5]に基づいた，再帰を含むプログラムに対するパラメトリック実行時間解析手法を提案する.提案手法は，以下の手順で構成される：

1. プログラムソースから，プログラムの各文の実行順の可能性を示す文脈自由文法を生成する
2. 得られた文脈自由文法を，各文の実行回数が等しくなるような正規表現に変換する
3. 得られた正規表現から，プログラムの実行時間をパラメータに関する式で導出する

また，本研究では解析対象のプログラムソースから得られた文脈自由文法を入力とし，その文脈自由文法を正規表現に変換して，最悪実行時間を出力するツールを試作する．このツールを用いていくつかの例題プログラムを対象にパラメータ付き最悪実行時間を導出し，提案手法の有効性を評価する.

2. 問題定義

2.1. 解析対象とするプログラム

本研究で扱うパラメトリック実行時間解析とは，入力としてプログラムの総実行時間を個々の手続き文の実行時間，およびプログラムの再帰回数やループ文の繰り返し回数に関する式の形で出力するものである．C 言語での再帰を含むプログラム例を図 1に示す.

```
1: Fact(N) {
2:   asm volatile ("marker=N");
3:   if(N==0) {
4:     return 1;
5:   } else {
6:     return N*Fact(N-1);
7:   }
8: }
```

図 1: C 言語でのプログラム例(fact.c)

図 1のプログラムのように，プログラムソースコードの各関数，またはループ文の内部に対して，明示的に対応付けするために，プログラムの設計者が手動でインラインアセンブラ構文を用いて“marker=X”(X はパラメータ変数名，または定数)という文字列を，コンパイル後のアセンブラコードの対応する箇所に埋め込まれるように挿入する.この文字列をマーカーと呼ぶ.マーカーに指定したパラメータ変数(もしくは定数)X

のことを以下，マーカー変数と呼ぶ.マーカーの位置のコードの繰り返し回数がマーカー変数に対応付けられているものとする．

入力となる解析対象のプログラムとは，図 1のような，再帰が書ける任意の手続き型プログラムで，繰り返し回数の最大値が有限であるようなプログラムとする．

2.2. 実行環境情報

プログラムの各文の実行時間は，CPU 速度など，実行環境に依存する．そこで，本研究では解析対象のプログラムを構成する各文（単位動作）の最悪実行時間があらかじめ与えられているとする．本研究では特に現実のプログラムコードを対象とし，コンパイラの最適化オプションに依存しない実行時間の見積もりを得るため，プログラムソースをコンパイルした後のアセンブラコード（ネイティブコード）を解析対象とし，プロセッサの動作周波数 f，および，各命令語の最悪実行サイクル数 w_j $(j=1,\ldots,p)$ (p は命令語の総数)をパラメータ変数，もしくは定数で与えるものとする．以下，f や w_j などのパラメータを実行環境パラメータと呼ぶ．例えば， ARM 社の組込み用 32 ビット RISC プロセッサ ARM7TDMI[6]の各命令語の実行サイクル数(一部)は図 2に示す通りである．

命令語	実行時間（CPUサイクル数）
レジスタ間データ処理	S
メモリロード命令（単一レジスタ）	$S+N+I$
メモリロード命令（複数レジスタ）	$nS+N+I$
メモリストア命令（単一レジスタ）	$2N$
分岐命令	$2S+N$
乗算命令	$S+mI(m\in\{1, \ldots, 4\})$

I：内部データ処理サイクル数(レジスタ間転送のみ)

S：逐次メモリアクセスサイクル数

N：非逐次メモリアクセスサイクル数

n：転送メモリワード数

m：乗数のワード幅（単位：バイト）

図 2: ARM7TDMI プロセッサの命令実行サイクル数 ([6]より抜粋)

2.3. パラメトリック実行時間解析問題

パラメトリック実行時間解析問題とは，2.1節で述べた方法でマーカー変数（又は定数）x_i $(i=1,\ldots,k)$ (k はマーカーの総数)によって注釈付けられたプログラムソースコードと，そのプログラムソースコードの実行環境情報，すなわち，プロセッサ動作周波数 f および各命令語の最悪実行サイクル数 w_j $(j=1,\ldots,p)$が与えられたときに，パラメータ付き最悪実行時間を表す式を導出することである.ここで，パラメータ付き最悪実行時間とは，入力となるプログラムを，実行環境情報で指定した実行環境で，マーカー位置の繰り返し回数が指定した回数となるように実行したときの実行時間の最悪値の見積もりを，各マーカー変数 xi，実行環境パラメータ f, w_j の関数の形で求めたものである．例えば，2.2節の ARM プロセッサの例では，各命令語の最悪実行サイクル数 w_j はパラメータ S,N,I の関数となるので，パラメータ付き最悪実行時間は，例えば$(4S + (2N + I) \times X)/f$ (X は繰り返し回数パラメータ)のように表現される． このパラメトリック実行時間解析問題の概要を図 3に示す．

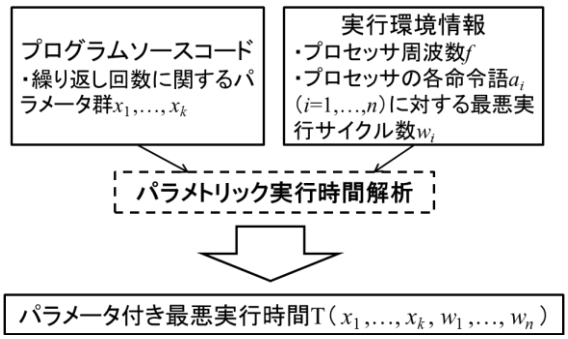

図 3：パラメトリック実行時間解析問題の概要

3. 提案手法

3.1. 提案手法の流れ

提案手法の説明として，まず，プログラムと文脈自由文法の対応について説明する．次に，文脈自由文法から正規表現への変換手法を説明し，最後に正規表現から実行時間を導出する方法を説明する．

3.2. 文脈自由文法の生成

文脈自由文法[7]は，左辺が一つの非終端記号で，右辺が終端記号と非終端記号からなる長さ 0 以上の系列であるような生成規則の集合，および開始記号の指定（非終端記号から一つ選択）からなるものである．文脈自由文法の詳細な定義は紙面の制約により割愛する．終端記号の集合を Σ とする．左辺が等しい２つの生成規則 $A \rightarrow w$, $A \rightarrow w'$を $A \rightarrow w \mid w'$と略記する.3 つ以上の場合も同様とする.また，特に断りのない限り，生成規則において英小文字および数字は終端記号，英大文字で始まる記号列は非終端記号を表すものとする．

解析対象となるプログラムソースコードから，文脈自由文法を生成する際，次のような方針で変換する．

・生成する語が，プログラムにおける文の実行順の任意の可能性を表すように変換する

・プログラムの逐次実行は「語の連接」に対応させる．

・if 文などの分岐は「｜」に対応させる

・関数定義は，関数に対応する非終端記号を導入し，生成規則の左辺にその非終端記号，右辺に関数の本体を表す語を記述することで表現する．

・関数呼び出しは，関数を非終端記号に対応させ，当該関数を呼び出している位置にその非終端記号を記述することで表現する．

・繰り返し文やジャンプ命令などは，等価な関数呼び

出しに変換する.

例えば図 1 のプログラムの関数 Fact を非終端記号 F とし，3 行目の if 文を a，4 行目を b，7 行目を c という実行環境情報から得られたパラメータ変数，または定数とし，上記の方針に従って変換すると，

$$F \rightarrow ab \mid acF \qquad (1)$$

となる.

3.3. 正規表現への変換

正規表現[7]は，以下の（a）～（f）の規則で定義されるもの全体である：

（a）記号 ε，ϕ は正規表現である

（b）$a \in \Sigma$ であるとき，a は正規表現である

（c）E と F が正規表現ならば，$(E+F)$も正規表現である

（d）E と F が正規表現ならば，$(E \cdot F)$も正規表現である.

（e）E が正規表現ならば$(E)^*$も正規表現である

（f）（a）～（e）までで定義されるもののみが正規表現である.

正規表現の意味は，Σ上の語集合への対応関係により定義される．正規表現 E,F に対応する語集合を [E],[F]とすると，$[\varepsilon]=\{\varepsilon\}$, $[\phi]=\phi$, $[E+F]=[E] \cup [F]$, $[E \cdot F]=[E] \cdot [F]$ (右辺の「・」は語集合の連接演算), $[(E)^*]=[E]^*$と定義される．正規表現 $E \cdot F$ は，「・」を省略して EF と書いても同じ意味であるとする.

文脈自由文法で生成される言語で正規表現では表せないものが存在することが知られている[7]. 従って，一般には任意の文脈自由言語に対してそれを表現する正規表現を求めることはできない.しかし，語の各記号の順序を入れ替えた語を等価とみなした場合，文脈自由文法が表す言語と等価な言語を表現する正規表現を求めることができる. 例えば「$aaabbb$」と「$ababab$」は異なる語である．しかし，「a」と「b」がそれぞれ単一で実行時間を表すとすると，2 つの語が表す実行時間は，どちらも「ab」の三つ分で等しいとみなせる．一般に文脈自由言語$\{a^n b^n \mid n>0\}$は正規言語ではないが，記号の順序を入れ替えても等価とみなすことにより，等価な正規表現$(ab)^*$を得ることができる.

そのような文脈自由文法から正規表現への変換は，可換 Kleene 代数における一般化 Parikh の定理[5]を用いることで行える.（可換）Kleene 代数は，ϕ を 0（加法単位元），ε を 1（乗法単位元）, +を加法('+'), ・を乗法('・'), *を*に対応させることで正規表現を代数で表現したものであり，形式的には以下のように定義される[5]：

[定義 1] Kleene 代数とは 7 つ組(Σ，S，+，・，*，0，1)で，任意の p，$q \in \Sigma \cup S$ に対して加法('+')に関してべき等（すなわち，$p+p=p$）であり，加法('+')および乗法('・')に関して半環('+'に関して逆元のない環)を成し，以下の公理を満たすものと定義する.

［公理 1］$1 + pp^* = p^*$

［公理 2］$1 + p^*p = p^*$

［公理 3］$q + pr \leqq r \rightarrow p^*q \leqq r$

［公理 4］$q + rp \leqq r \rightarrow qp^* \leqq r$

ただし，「≦」は以下のように定義する.

$$p \leqq q \Leftrightarrow p + q = q$$

可換 Kleene 代数とは，乗法「・」が可換であるような Kleene 代数であると定義する. □

例として，図 1 のプログラムから生成した文脈自由文法(1)を文献[5]の手法に基づいて正規表現に変換する手順を示す. まず, 文法の各生成規則から可換 Kleene 代数の不等式へと変換する. 可換 Kleene 代数の不等式への変換を下記に示す.

・各非終端記号を変数(x，y，z など)に変換

・「｜」は和(+)に，連接は積(・)に変換

・各終端記号はそのまま定数として扱う

・ループ（for 文など）は，$(\ )^*$に変換する(*は繰り返し回数とする)

・空語(ε)は定数 1 に，空集合(ϕ)は定数 0 に変換

・生成規則（左辺）→（右辺）を不等式

（右辺）≦（左辺）に変換

これに従って(1)式を変換すると，以下のようになる.

$$ab + acx \leqq x \qquad (2)$$

次に得られた可換 Kleene 代数上の連立不等式系の最小解を求める. この最小解は可換 Kleene 代数において元の文脈自由文法が生成する言語と等価となる[5]. この最小解を可換 Kleene 代数における一般化 Parikh の定理に基づいて導出する．この定理は以下のように述べられる：

［定理 1（一般化 Parikh の定理[5]）］ 可換 Kleene 代数の不等式 $f(x) \leqq x$ の最小解は $f'(f(0))^* \cdot f(0)$で表せる. ただし，$f(x)$を可換 Kleene 代数上の x に関する任意の多項式とし，$f'(x)$は以下を満たす $f(x)$の変形操作(f の「微分」と呼ばれる)とする.

（a）$f'(x)=1$

（b）$f'(a)=0$

（c）$f'(1)=0$

（d）$(f(x) \cdot g(y))' = f(x)\, g'(y) + g(y)\, f'(x)$

（e）$(f(x)+g(y))' = f'(x) + g'(y)$

（f）$(f(x)^*)' = f(x)^* \cdot f(x)'$ □

定理 1 を用いて式(2)の最小解を求めると，

$$(ac)^* \cdot ab \qquad (3)$$

となり，これが図 1 のプログラムと各文の実行回数が等しくなるような正規表現となる.

3.4. 最悪実行時間の導出

最悪実行時間は正規表現から導出する．正規表現 E に対応する最悪実行時間を $ET(E)$とすると, $ET(0) = \infty$ (停止しないプログラムに対応), $ET(1) = 0$, $ET(E \cdot F) =$

$ET(E)+ET(F),\ ET(E+F) = max(ET(E), ET(F)),\ ET(E^*) = ET(E)\times m(E)$ (ただし，$m(E)$は正規表現 E 中で*演算子のスコープ外に現れるマーカー変数[1])という規則に従って導出する．例えば図 1の関数 Fact の呼び出し回数を表すマーカー変数を N とし，式(3)の正規表現から実行時間を求めると，以下のようになる．

$$ET(\text{Fact})=(ET(a)+ET(c))\times N+ET(a)+ET(b) \qquad (4)$$

4. 実装

4.1. 試作したツールの概要

本研究では解析対象の再帰を含むプログラムソースコードから得られた文脈自由文法を入力とし，可換 Kleene 代数の方程式の解を求めて，パラメータ付き最悪実行時間を表す式を出力するパラメトリック実行解析ツールを作成した．図 4に解析対象の最悪実行時間を表す式を導出するツールの概要を示す．

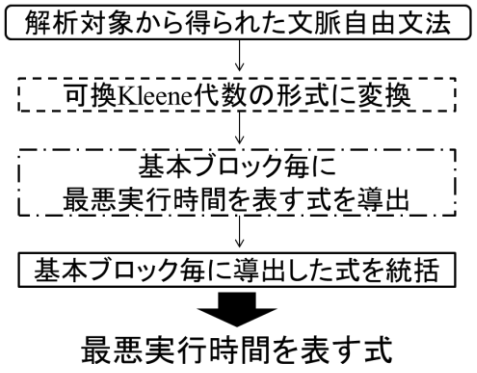

図 4：ツールの概要

4.2. 入力形式

試作したツールは次のような文脈自由文法 G を入力とする.実行環境情報は2.2節で例示した ARM7TDMI プロセッサの情報を用いる[2].G は非終端記号を S_i (i は先頭から i 番目の命令語に対応)とした文脈自由文法 $S_i \rightarrow w$の集合である．ただし，w は可換 Kleene 代数の記法で次のように記述されるものとする．例えば，図 1 の文脈自由文法(1)を入力とすると，

$$S1 \rightarrow a\cdot b + a\cdot c\cdot S1 \qquad (5)$$

となる．実際には解析対象プログラムをコンパイルした後のアセンブラコードの各命令語について次のような生成規則を対応させる[3].まず各命令語に対して終端記号を対応付ける.次に，解析対象プログラムの i 番目の命令語が分岐以外の通常の命令語 a ならば，生成規則 $S_i \rightarrow a\cdot S_{i+1}$ に対応させる．特に a がマーカー$m(X)$ (X はマーカー変数名)であっても同様に，生成規則 $S_i \rightarrow m(X)\cdot S_{i+1}$ に対応させる.i 番目の命令語が無条件分岐 b ならば生成規則 $S_i \rightarrow b\cdot S_j$ (j は分岐先の命令語の番号),条件分岐 c ならば $S_i \rightarrow c\cdot S_j + c\cdot S_{i+1}$、無条件コール命令 d ならば $S_i \rightarrow d\cdot S_j\cdot S_{i+1}$ と変換する（条件コール命令なども同様に変換可能）．最後に，開始記号は S_1 とする.このように得られた文脈自由文法を入力とし，二分木のデータ構造に変換する.この変換部分の実装は，構文規則に基づいてパーサ(ソースコードの構文解析を行うコンパイラの一部分)を生成する yacc (Yet Another Compiler-Compiler)，および，字句解析を行う lex (Lexical analyser generator)を用いて行った．

4.3. 最悪実行時間を表す式の導出

文献[6]より，文脈自由文法の生成規則 $x\rightarrow f(x)$に対応する可換 Kleene 代数の不等式 $f(x)\leqq x$ の最小解は，

$$f'(f(0))^*\cdot f(0) \qquad (6)$$

で求められることが知られている．例えば，式(5)の生成規則に式(6)を適用すると，$S1$ を表す正規表現を以下のように求めることができる．

$$S1 = (a\cdot b)^*\cdot a\cdot c \qquad (7)$$

本研究ではこの求めた解により，一般にプログラム及び実行環境のパラメータ付き最悪実行時間の見積もりを導出する．この解を導出する操作を以下に示す．

1. $f(x)$の変数 x に 0 を代入し，$f(0)$を求める
2. $f(x)$を変数 x で微分し，$f'(x)$を求める
3. $f'(x)$の変数 x に $f(0)$を代入し，$f'(f(0))$を求める

微分を求める操作については．定理 1 に基づいて行う．なお，複数の変数を含む式が入力となる場合は，そのうち 1 つの変数に着目し，その他の変数を定数とみなして定理 1 を適用することにより変数を 1 つずつ消去する.例えば，

$$S2 = a\cdot S2 + b\cdot S1 \qquad (8)$$

のような場合は，変数 S2 に着目し，変数 S1 を定数とみなして定理 1 を適用し解を求めると，

$$S2 = a^*\cdot b\cdot S1 \qquad (9)$$

という解が得られる．この $S2$ の解である(9)式の右辺には変数 $S2$ が現れることはなく，また，変数 $S2$ を持つ他の不等式に代入すれば，入力内容全体から $S2$ という変数を消去できる．例えば，式(9)にはまだ $S1$ という変数が残っているが，式(9)に式(7)を代入すれば，

$$S2 = a^*\cdot b\cdot (a\cdot c)^*\cdot a\cdot b \qquad (10)$$

という解を得ることができる.このような手順により，相互再帰など任意の再帰構造を持つプログラムに対して解を得ることができる．

4.4. 分割統治法による効率化

複雑な式になると計算量が増えてしまい，ツールを用いる環境によっては導出が困難になる可能性がある．そこで，本研究では基本ブロック毎にパラメータ付き

---

[1] 入れ子になっている場合を除き，一つの繰り返し中にマーカーの指定はただ一つしか存在しないと仮定している．入れ子の繰り返し構造の場合は，内側の繰り返し回数に対応するマーカーは内側の*演算子のスコープ内に存在する．

[2] 他の実行環境にも容易に対応可能である．

[3] ただし本ツールは，アセンブラコードから変換した文法のみならず，ユーザが任意に入力した文脈自由文法を処理可能である．

最悪実行時間を表す式を導出し，その後に導出したすべての式を統括する分割統治法で導出の簡単化を図る．

ここで基本ブロックとは，コンピュータにおいて一つの入り口と一つの出口を持ち，内部に分岐を含まないプログラムソースコードを指す．基本ブロックの終了は，他のプログラムソースコードへのジャンプ命令である．以下に，この基本ブロック毎に分割してパラメータ付き最悪事項時間を表す式を導出する手順をまとめる：

1. 入力を可換 Kleene 代数の形式に変換する．
2. 変換された入力を基本ブロック毎に分割する．
3. 基本ブロック毎にパラメータ付き最悪実行時間を表す式を導出する．
4. 各基本ブロックを単位動作とみなした文脈自由文法を構成し，各基本ブロックの最悪実行時間を未定パラメータとしたパラメトリック最悪実行時間の式を導出する．
5. 4.で求めた式に3.で求めた各基本ブロックの最悪実行時間を表す式を代入し，簡約化する．

5.で簡約化が終わると，パラメータ付き最悪実行時間の式が $max\{e_1,\ldots,e_m\}$という形で得られる．ただし，m は実行時間が最悪になり得るプログラムの実行経路の総数である．これらの式 $e_1,\ldots,e_m$ をパラメトリック最悪実行時間の候補と呼ぶ．

5. 実験

5.1. 実験目的･方法

図 1で示した再帰を含む例題 fact.c(命令語数 39), および文献[8]で用いられた，カメラで撮影して得られた画像から重心を計算する画像処理プログラムの例題 camera_example.c(命令語数 164)を対象とし，適用実験を行った. 実験環境は, CPU : Intel Core2 Duo 2.53GHz, OS : Windows XP Professional, メモリ : 1.98GB である．この下で，camera_example.c と fact.c のパラメータ付き最悪実行時間を導出し，結果の正しさを評価する．また，camera_example.c のサイズの異なるいくつかのコード断片について導出することにより，プログラムサイズに対する導出時間の変化を評価する．

5.2. 実験結果

fact.c では再帰呼び出しの繰り返し回数を示すマーカー変数 N の 1 次式で，パラメータ付き最悪実行時間が正しく導出され，導出時間は 0.2 秒かからなかった．camera_example.c では画像処理の繰り返し回数を示す縦画素数と横画素数のマーカー変数の 2 次式で，パラメータ付き最悪実行時間が正しく導出され，導出時間は 1.5 秒かからなかった．よって，解析の導出時間は小規模のプログラムに対しては十分高速であるといえる．次に、camera_example.c のコンパイル後のコードについて, 先頭から n 語まで(n = 66,88,118,141,152,164) のコード断片について解析を行い，導出時間の変化を評価した．その結果を図 5に示す．図 5の 66 語～130 語までの解析結果では最悪実行時間の候補が 2 つ導出でき, 141 語～164 語の解析結果では候補が 4 つ導出できた．図 5より，最悪実行時間の候補数が導出時間に差をつけたと言える．また，候補が 2 つの時と 4 つの時では，それぞれ導出時間がほぼ一定であるため，導出時間は最悪実行時間の候補数に関係すると言える．

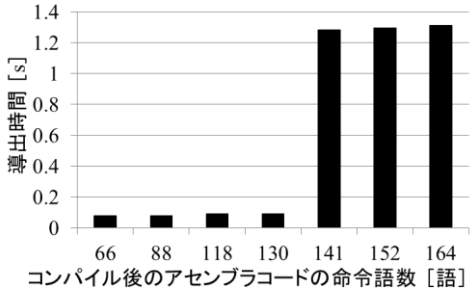

図 5：実験結果

6. あとがき

本研究では，一般化 Parikh の定理に基づいた再帰プログラムのパラメトリック実行時間解析手法を提案し，提案手法を実現するツールを試作した．また，試作したツールにより，いくつかの例題に対して有効性を評価した．今後の課題としては，より大規模な例題に対して提案手法の評価を行うことが考えられる．

参考文献

[1] N. G. Leveson and K. A. Weiss: “Making embedded software reuse practical and safe”, Proc. of the 12th ACM SIGSOFT Int. Symp. on Foundations of Software Engineering (FSE 2004) ,pp.171-178, 2004.

[2] P. Puschner and A. Burns: “A review of worst-case execution-time analysis”, The Int. Journal of Time-Critical Computing Systems, 18, pp.115-128, 2000.

[3] J. Engblom, A. Ermedahl, M. Sjodin, J. Gustafsson and H. Hansson: “Worst-case execution-time analysis for embedded real-time systems”, Int. Journal of Software Tools for Technology Transfer, 4, pp.437-455, 2003.

[4] 河井敏宏,中田明夫:「実時間ソフトウェア再利用のためのパラメトリック実行時間解析の一手法」,信学技報 SS,Vol.107,No.505,pp.97-102, 2008.

[5] W. Hopkins and C. Kozen: “Parikh's theorem in Commutative Kleene Algebra”, Proc. of the 14th IEEE Symp. on Logic In Computer Science (LICS 1999), pp.394-401, 1999.

[6] ARM Ltd.: “ARM7TDMI Technical Reference Manual”, Revision r4p1, http://infocenter.arm.com/help/topic/com.arm.doc.ddi0210c/, 2004.

[7] 都倉信樹:「オートマトンと形式言語」, pp. 47-128, 昭晃堂, 2002.