１．取り組む研究について

（以下の各項目について簡潔に記載下さい。本項目のページ数：5 ページ以内）

（１）取り組む研究のタイトル

・本欄には、本提案に係る研究のタイトルを日本語の場合は 30 字以内で、英語の場合は 20 ワード以内で記載して下さい。

型推論をそなえた言語による UNIX ライク OS の研究開発と実証実験

（２）研究の概要

・本欄には、本提案に係る概要を日本語の場合は 100 字程度で、英語の場合は 50 ワード程度で記載して下さい。

型推論を持つ言語による安全な設計はアプリケーションのみに適用できるものではなく UNIX ライク OS の kernel 本体にも適用することができ、さらに既存のどのソフトウェア領域にも適用できることを実証する。

（３）研究計画等

・本欄には、研究目的、研究計画・方法・場所、研究成果、業績、その他アピールしたいことについて自由に記述して下さい。また、必要に応じて図表を入れることも可とします。

【研究の経緯】

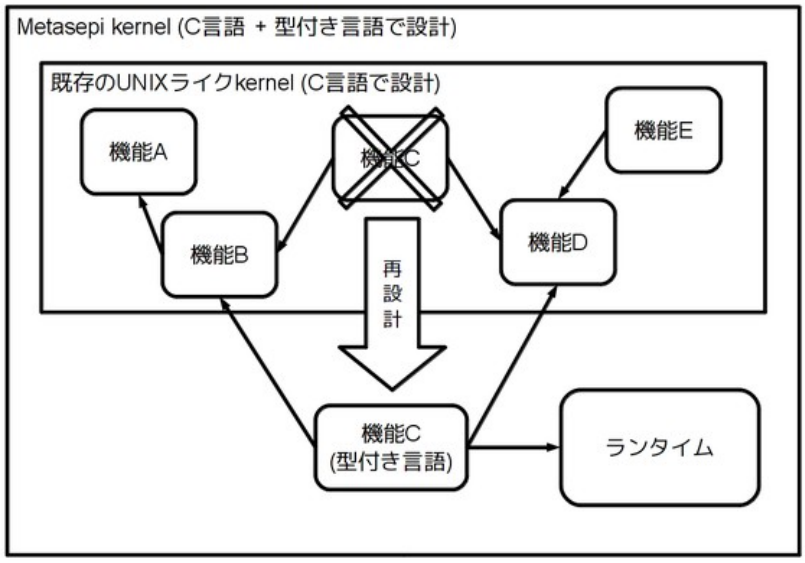
私は 10 年間大規模組み込み開発を経験してきました。その開発を通じて機能開発はもちろんですが、それよりも多くの不具合/市場障害を経験しました。私はこの不具合とそれによる開発者の不幸の根本原因がいったい何なのかこの数年間追い求めてきました。そしてこれらの原因はやはり製品を設計した後の実行時エラーにあると考えるようになりました。OS ではなくアプリケーションドメインでは型推論をそなえた言語による設計が実行時エラーの削減に一定の成果をあげています。しかし本当に安全な設計が必要なのはアプリケーションではなく OS のような基盤ソフトウェアであると考えます。アプリケーションに動作異常ががあった場合最悪のケースではセグメンテーションフォールトによるプロセスの実行停止で済みます。しかし OS に動作異常が起きた場合容易に機器全体の停止が起きてしまいます。アプリケーションとは異なり、OS の設計には細心注意が求められます。

にもかかわらず現在でも OS の設計には旧来の C 言語が主に使われています。C 言語による設計は危険ですが、膨大な工数をかけることで品質を担保しているのが現在の製品開発の姿です。しかしそれでは単なる人海戦術です。C 言語によって OS を記述する事は 1970 年台にはそれに代わるものがその当時存在しなかったという意味では最良のものでした。しかしその後怒涛の勢いで発達したコンパイル技術や、とりわけ型理論の整備の成果を考えれば、現時点においても C 言語による OS 開発が最高のものだとは言い難い状況です。

それではなぜ OS、特に kernel について型推論をそなえた言語による設計が行なわれず、C 言語による設計が主なのでしょうか。型推論をそなえた言語が持つガーベッジコレクタと OS 設計との相性が悪いなどの理由がありますが、実際には型推論をそなえた言語によって kernel を設計する試みはいくつもあります。

(参考文献 1,2,3,4) ところがこれらの OS は実験的な試みで終わってしまっています。これらの OS の上で Firefox のようなデスクトップ上で日常的に使うアプリケーションは動作しません。またこれらのソースコードを読むかぎりでは近代的な OS に見られる抽象化がなされていないため、移植性と拡張性に乏しいと考えられます。この状態でも限定用途の OS として使うことは可能でしょうし、またこの実装から実際のデスクトップ OS として使えるまでに成長させることも不可能ではないかもしれません。しかし現代の UNIX OS は 1977 年から脈々と拡張性と抽象化のしくみを構築してきました。まずは既存 OS の設計を真似て動作可能な OS を安全な手法で設計し、その後で独自の機能を付加していった方が良いと考えられます。

そこで私は既存の C 言語で記述された設計を型推論をそなえた言語で少しずつ設計置換する手法を考案しました。(右図)この設計手法を arafura デザイン(参考文献 5)と呼びます。まず C 言語で設計された既存のソフトウェアを用意し、その一部の機能に焦点をしぼり型付き言語によって機能を置換します。arafura デザインのソフトウェアはコンパイル可能であり、また実行可能ですこの実行可能な状態を維持しながら少しずつ C 言語の設計に強い型をつけることで、最終的には全てソースコードを型推論をそなえた言語によって書き直すことができるはずです。



残念ながら arafura デザインを行なえるような既存の型推論をそなえた言語のコンパイラは、インターネット上に公開された情報からは見つけることができませんでした。そこで、arafura デザインに適したコンパイラに近い機能を持つコンパイラに改造を加えることにしました。

表: “hoge”とコンソールに印字するプログラムの特性

	バイナリサイズ(バイト)	未解決シンボル数	依存ライブラリ数
GHC 7.4.1	797228	144	9
jhc 0.8.0	21248	20	3
OCaml 4.00.1	183348	84	5
MLton 2010608	170061	71	5
SML# 1.2.0	813460	134	7

上記の表は文字列を印字するだけのプログラムを著名なオープンソースの型推論をそなえたコンパイラの吐き出す実行バイナリの特性をまとめたものです。組み込み用途なので「バイナリサイズ」は小さい方が良いのですが、それよりももっと重要なことは「未解決シンボル数」と「依存ライブラリ数」です。arafura デザインをアプリケーションに適用する場合にはこの数は問題にはなりませんが、kernel のような既存ライブラリに頼れないソフトウェアを開発する際には実行バイナリが依存する機能の数が少なければ少ないほど楽ができます。もし依存ライブラリ数が多い場合には既存の C 言語プログラムに左記ライブラリと同等の機能をまず作りこまなければいけません。さらに、その作りこんだ C 言語の設計を強い型を使って再設計することは arafura デザインをもってしても困難になります。上記の表から明確に jhc がずばぬけた良い特性を持っていることがわかります。ところがこの jhc コンパイラについての論文は一件もなく、またこのコンパイラについて調査している研究者はインターネット上での公開情報からは見当りませんでした。私はこの jhc を元にして arafura デザインを行なえるコンパイラを作るプロジェクト Ajhc(参考文献 6)を開始しました。2013 年 05 月 05 日現在まで 2 回の正式リリースをしています。(参考文献 7,8) また Ajhc での機能追加の一部は jhc 本体に取り込まれています。arafura デザインがアプリケーションだけではなく OS ドメインについても適用可能なことを実証するため、まずは C 言語で記述された NetBSD bootloader と STM32 マイコン(メモリ 40kB)のデモの一部を Haskell 言語で再設計しました。(参考文献 5,9,10) もちろん一部だけ設計置換しても全体を型によって再設計できると実証できたことにはなりません。さらなる研究開発が必要で、そこにははるかに大きい課題が待っているはずです。

【研究目的】

C 言語で記述されたソフトウェアを型推論をそなえた言語で少しずつ設計置換する(arafura デザイン)ことが OS のような低レベルプログラミングでも可能であることを実証します。さらにこの設計置換手法がコンピューターアーキティクチャの全てのドメインに適用可能であることを実証します。

【研究計画と方法】

1. 小さなマイコンで動く C 言語デモの arafura デザインによる設計置換

STM32 マイコンを使った簡単なデモを Haskell 言語によって設計置換します。現状でも main 関数より上位の部分については Haskell 化できていますが、時間経過の待ち合わせ、デバイスの初期化、割り込み処理については Haskell 言語によって設計置換できていません。特に割り込み処理については現在の Ajhc コンパイラで実現することは不可能です。割り込みコンテキストと通常コンテキストのデータ受け渡し方法や、GC 中の割り込み、割り込みコンテキストにおけるヒープの取り扱いなど難題が多くあります。

2. 小さな OS(FreeRTOS など)の arafura デザインによる設計置換

1 では OS なしでマイコンを動かしていました。このままでは設計対象の抽象度が低いため、実際に使われている OS を設計置換します。しかしいきなり UNIX ライク kernel を置換対象にするとコンパイラの機能不足分が大きすぎてプロジェクトが停滞する可能性が高いです。そこでより小さな OS である FreeRTOS を例題にして設計置換を行ないます。FreeRTOS は 1 で対象にした STM32 マイコン上での動作も可能なため、デバッグ環境等は共有することができます。1 で割り込み処理に抜本的な解決ができなかった場合に

はこの段階でも割り込み処理の置換手法が大きな課題になるでしょう。FreeRTOS を設計置換できた場合にはヘリコプター制御などのアプリケーションを作成することでリアルタイム応答性のアピールをすることができるようになります。(参考文献 11)

3. UNIX ライク kernel(NetBSD など)の arafura デザインによる設計置換

FreeRTOS の大部分を Haskell 言語によって設計置換できれば UNIX ライク kernel を設計置換することが可能になるでしょう。置換対象としては NetBSD kernel を選択します。この設計置換された kernel を Metasepi(参考文献 12)と呼びます。Metasepi OS の上で Metasepi 自身の開発が可能になることがこのプロジェクトの最初のゴールです。私は NetBSD kernel を使った製品開発経験がありますが、それでもその全てのソフトウェア部品を熟知しているわけではありません。NetBSD kernel そのものに関する勉強会の運営(参考文献 13)と、NetBSD kernel マニュアルの翻訳(参考文献 14)を通じて詳細理解を行ない、設計置換が容易な部品を発掘します。さらに左記活動を通じて Metasepi 開発者の勧誘を行ない、工数確保をめざします。また Haskell 言語はアプリケーションドメイン向けに設計された面が大きいため、OS のようなシステム設計において Haskell 言語の既存の機構をどのように解釈すべきか検討します。

4. 上記 1,2,3 を実現可能なコンパイラの研究開発

型推論を持つ言語として Haskell 言語と Ajhc コンパイラにはこだわりませんが、現状では Ajhc 以外に有効な解が見当らないためこのセットで研究開発を行ないます。上記 1,2,3 以外に行なうべき研究課題として Ajhc コンパイラの品質維持が挙げられます。Ajhc の元になった jhc のユーザ数はとても少なく、また Ajhc では実験的な機能追加を行なうため現状の品質を維持することが課題になります。さらに Ajhc は Haskell 言語コンパイラのデファクトスタンダードである GHC と比較して機能が不足しているため、放置していても利用者は増加しません。そこで以下のような施策を実施します。

- Ajhc へ Haskell Platform(Haskell の基本ライブラリ群)を移植します。移植に際して Ajhc に不足している機能をあぶりだすことができます。
- 継続的インテグレーションにおける回帰テストの実施します。機能追加による不具合の発生が回帰テストによって自動的に検出されます。この回帰テストは既に実施しています。(参考文献 15)
- Haskell コンパイラソースコードリーディング勉強会の開催します。Ajhc の利用者と開発者を発掘できる可能性があります。これまで 2 回開催しています。(参考文献 16,17)
- Ajhc をインターネット上のサーバに配置して利用者は Web ブラウザだけを用意すれば Ajhc による開発を開始できるようにします。Ajhc の利用者拡大が見込める上、サーバ上でのコンパイルエラーを吸い上げることで Ajhc の品質向上にも貢献できます。
- 研究成果を ESEC(参考文献 18)や OSC(参考文献 19)などの展示会に出展します。Ajhc コンパイラについて共同研究先を発掘できる可能性があります。
- 研究成果について論文化します。

また Ajhc のランタイムは C 言語によって記述されていてその量は 3000 行程度ですが、今後の機能追加で膨張する可能性があります。設計対象を Haskell 言語化できたとしても Ajhc のランタイムに巨大な C 言語設計があっては問題の解決になりません。そこで以下のような施策を実施します。

- ライタイムの主要な部分はガーベッジコレクタであるため、その負荷を下げるために部分的なリージョン推論導入を検討します。 (jhc の過去のリリースではリージョン推論をそなえていたようです)
- ランタイムの一部をハードマクロ化できないか FPGA を使った協調設計を行ないます。 (ハードマクロ化の例: ガーベッジコレクタによるビットマーキングの 0 クリアをハードマクロ化してパフォーマンスを向上させる)
- ランタイムが肥大化するようであれば、ランタイムの品質を維持するために C 言語への形式手法の適用を検討します。

【場所】

日本国。

研究開発の状況によっては専門知識を持つ研究者との共同研究を行ないます。以下はその例です。

- フランス国立情報学自動制御研究所(OCaml コンパイラの開発元) http://www.inria.fr/
- Microsoft Research Cambridge(GHC コンパイラの開発元) http://research.microsoft.com/

【参考文献】

1. Funk (OCaml 製) http://home.gna.org/funk/
2. snowflake-os (OCaml 製) http://code.google.com/p/snowflake-os/
3. House (Haskell 製) http://programatica.cs.pdx.edu/House/
4. HaLVM (Haskell 製) http://corp.galois.com/halvm/
5. デザイン Arafura - Metasepi http://metasepi.org/posts/2013-01-09-design_arafura.html
6. Ajhc - arafura-jhc http://ajhc.metasepi.org/
7. Ajhc 0.8.0.2 Release http://www.haskell.org/pipermail/jhc/2013-March/001028.html
8. Ajhc 0.8.0.3 Release http://www.haskell.org/pipermail/jhc/2013-April/001047.html
9. Ajhc プロジェクトはじめよう http://metasepi.org/posts/2013-03-16-found_ajhc.html
10. ajhc/demo-cortex-m3 · GitHub https://github.com/ajhc/demo-cortex-m3
11. Crazyflie Nano Quadcopter Kit http://www.bitcraze.se/2013/02/pre-order-has-started/
12. Home - Metasepi http://metasepi.org/
13. Meeting0 · start-printf/wiki Wiki https://github.com/start-printf/wiki/wiki/Meeting0
14. netbsdman.masterq.net http://netbsdman.masterq.net/
15. Travis CI https://travis-ci.org/ajhc/ajhc
16. HaskellJP wiki - Workshop/ReadGHC/0 http://wiki.haskell.jp/Workshop/ReadGHC/0
17. HaskellJP wiki - Workshop/ReadGHC/1 http://wiki.haskell.jp/Workshop/ReadGHC/1
18. 組込みシステム開発技術展（ESEC) http://www.esec.jp/
19. OSPN - We Are Open!! http://www.ospn.jp/

2．研究業績について

・本項目のページ数：4 ページ以内

・本欄には、これまでに発表した論文、著書、産業財産権、招待講演等のうち、主要なものを選定し、各区分（論文、著書、産業財産権等の区分）毎に現在から順に発表年次を過去にさかのぼり、通し番号を付して記入してください。なお、学術誌へ投稿中の論文を記入する場合は、掲載が決定しているものに限ります。

・例えば発表論文の場合、論文名、著者名、掲載誌名、査読の有無、巻、最初と最後の頁、発表年（西暦）について記入してください。以上の各項目が記載されていれば、項目の順序を入れ替えても可とします。

・著者名が多数にわたる場合は、主な著者を数名記入し以下を省略（省略する場合、その員数と、掲載されている順番を○番目と記入）しても可とします。なお、応募者には下線を付して下さい。

【著書】

1. 「Lighter than Light」 岡部究. λカ娘 4 巻. 査読無し. 15-34 頁. 2012 年 12 月
2. 「RTS 海溝二万マイル」 岡部究. λカ娘 3 巻. 査読無し. 33-54 頁. 2012 年 08 月
3. 「Copilot への希望と絶望の相転移」 岡部究. λカ娘 2 巻. 査読無し. 7-21 頁. 2011 年 12 月
4. 「OS を侵略しなイカ？」 岡部究. λカ娘 1 巻. 査読無し. 35-39 頁. 2011 年 08 月
5. 「 Multimode Quartz Crystal Microbalance 」 GOKA Shigeyoshi, OKABE Kiwamu, WATANABE Yasuaki, and SEKIMOTO Hitoshi. Japanese journal of applied physics. 査読有り. 3073-3075 頁. 2000 年 05 月

【講演】

6. 「Haskell ではじめる Cortex-M3 組込みプログラミング」 オープンソースカンファレンス 2013 Tokyo/Spring / ライトニングトーク@Business Day. 2013 年 02 月
7. 「スタート低レイヤー #0」 オープンソースカンファレンス 2013 Tokyo/Spring / NetBSD のご紹介. 2013 年 02 月
8. 「What is Metasepi?」 秘密結社 Metasepi 作戦会議. 2013 年 02 月
9. 「Dive into RTS - another side」 GHC ソースコードリーディング勉強会 第 1 回. 2012 年 09 月
10. 「Emacs と Gloss でお絵描きしてみるよ」Haskell Day 2012. 2012 年 05 月
11. 「NetBSD man を翻訳しよう! (OSC2012 版)」 オープンソースカンファレンス 2012 Tokyo/Spring . 2012 年 03 月
12. 「GHC ソースコード読みのススメ」GHC ソースコードリーディング勉強会 第 0 回. 2012 年 02 月
13. 「Debian Loves Haskell」 Tokyo Linux Users Group (TLUG) Technical Meeting. 2011 年 11 月
14. 「Haskell と Debian の辛くて甘い関係」 第 81 回東京エリア Debian 勉強会. 2011 年 10 月
15. 「Cairo でほっかり GUI プログラミング」 第 0 回 スタート Haskell. 2011 年 07 月
16. 「Wiki 設置するなら gitit!」 第 6 回 qpstudy06. 2011 年 05 月

【オープンソース活動】

17. Ajhc Haskell コンパイラの開発 http://ajhc.metasepi.org/ 2013 年 03 月〜
18. Metasepi OS の研究開発 http://metasepi.org/ 2012 年 08 月〜
19. プレゼンテーションツール Carettah の開発 http://carettah.masterq.net/ 2011 年 07 月〜
20. NetBSD man ページ翻訳環境の整備 http://netbsdman.masterq.net/ 2011 年 04 月〜
21. Debian Maintainer による複数の Debian パッケージの管理 2009 年 09 月〜

3．研究者としての抱負について

（以下の各事項について簡潔に記載下さい。なお、本項目については主として第２次審査の面接において使用します。本項目のページ数は２ページ以内とします。なお２ページの範囲内であれば、必要に応じて各回答欄の大きさを調整いただいて構いません。）

（１）研究の理由を記してください。

・これまでのあなたのキャリアとプロジェクト終了以後のあなたの思い描くキャリアに触れつつ、あなたがプロジェクトに応募した理由を記載してください。

私は 10 年間大規模組み込み開発を経験してきました。その開発を通じて機能開発はもちろんですが、それよりも多くの不具合/市場障害を経験しました。私はこの不具合とそれによる開発者の不幸の根本原因がいったい何なのか、この数年間追い求めてきました。現在 OS のような基盤ソフトウェアの品質維持は人海戦術に頼っていると言わざるをえません。そのノウハウは工学的な解決をなされているのでしょうか？またこの問題を解決することに研究者は積極的ではありません。たしかに彼らが作る OS は新規性にあふれています。しかし実用からはかけはなれています。彼らの作った OS の上で Firefox や Emacs は動きません。日常用途として使わないソフトウェアはいずれ朽ちる運命にあります。

この研究状況を観察した結果、私は手をこまねいてただ待つだけではこの領域における研究開発は進まないと考えるようになりました。この領域における研究開発には以下の技能が必須です。

- プロジェクトモチベーションを理解するための大規模組み込み開発の実地経験
- UNIX のような近代的な OS に関する深い理解
- 論文ではなくソースコードを読解する体力
- 型推論をそなえた言語の利用経験

様々なコミュニティで Metasepi プロジェクトについて紹介をして共同研究できる先を探してみましたが、上記をみたす人材はなかなか見つかりません。「数年の間 Metasepi と Ajhc に関する研究開発はほぼ独自に行ない、研究成果をわかりやすい形でアピールし研究の裾野を広げる必要がある」というのが私の現在の結論です。

（２）あなたにとっての理想の研究者像とはどのようなものですか？

・あなたにとっての理想の研究者像を簡潔に記載して下さい。

目的のためには手段を選ばない研究者を理想としています。また学術的な通説(もしくは権威ある意見)についても検証が終わるまでは中立の立場を貫くようにしようと考えています。

Metasepi の開発にあたって既存のコンパイラのサーベイをしましたが、C 言語の設計を少しずつ型の強い言語で設計置換(arafura デザイン)できるようなコンパイラはなかなか見つかりませんでした。そんなおり論文が一件もなく研究者からの全く注目されていない jhc コンパイラこそが最も arafura デザインに近いコンパイラであることを jhc のソースコード解析からつかみました。

これからの Metasepi と Ajhc の開発においても学術的な通説を疑い、ありとあらゆる手段をつくすつもりです。例えば、ランタイムを持つ言語で OS を設計する際には通常の UNIX ライク kernel のように「割り込みハンドラ」を使うのではなく、「割り込み要因をポーリング」した方が良いとされています。これについても中立の立場を取る予定です。現状、前者による実装が困難なのであれば、前者を容易に実現可能なコンパ

イラを作れば良いのです。さらに現状ではAjhcのランタイムとコンパイルパイプラインはシンプルになっていますが、今後肥大化/複雑化する可能性があります。このような状況を制御するために形式手法が必要であれば、当該分野を学習します。
私は自分の専門領域の中にのみ注力することが目標達成のための最善手ではないと考ます。専門外にある分野を自分で理解し、自分の中にある解を発見しようと努めます。もちろん専門家の意見は尊重します。しかしプロジェクトにおける解は他者である専門家の中にあるのではなく、ステークホルダーである自分の中にしかないと硬く信じています。

（３）一人の研究者として、現在、世界が抱える諸課題の解決に向けてどのような貢献が出来るとお考えですか？

・どのような課題でも結構ですので、簡潔に記載下さい。

Linux kernelを代表とするフリーUNIX OSは世界を変えました。20世紀に発明されたこのOSは世界のありとあらゆる機器を動かす基盤ソフトウェアになりました。20世紀において携帯電話がフリーUNIXで動く(Android、iPhone)と誰が想像しえたでしょうか？電話交換機のようなリアルタイムが要求されなおかつミッションクリティカルな機器がLinuxによって制御され、旧来のμITRONによる設計がお払い箱になると誰が想像しえたでしょうか？さらには現代においてサーバOSの代名詞はLinuxです。しかも2013年の現在もその適用領域は拡大しています。
しかしこのフリーUNIX OSの信頼性の確保は21世紀の現代でも難題であり、「オープンソース開発」と名前のつけられた人海戦術に頼っています。その問題の多くはソフトウェアの実行時エラーを人力でデバッグしていることに起因すると考えます。確かにこの「オープンソース開発」手法の上で開発され機能については人海戦術によって信頼性を確保することはできるかもしれません。しかしその主流開発の外での機能追加において信頼性の確保を行なうにはより多くの人海戦術のための人材が必要です。また、当然人海戦術のために世界中の有能なソフトウェア開発者の頭脳が日々浪費されています。このような開発スタイルは文明的なのでしょうか？
また、フリーUNIX OSはRTOSのような小規模なOSよりも機能と設計の規模が大きいため、人工衛星の制御のような領域にはまだ踏み込めていません。ハードリアルタイム制御についても同様で、これらの領域はVxWorksやμITRONのような機能の不足したOSに占有されています。
Metasepiはこれらの状況を打開します。強い型による実行時エラーの防止がLinux kernelのような基盤ソフトウェアの領域でも有効であると実証します。また、左記設計に必要な強い型を持つ言語のコンパイラを世界に提供します。このコンパイラを使ってMetasepiではない別のソフトウェアを第三者が作ることも可能になるでしょう。さらにこのコンパイラを用いることでほとんどのソフトウェア領域に強い型による設計を適用できることを実証します。また、型による設計によって設計の見通しが良くなり高信頼性が要求される領域にもフリーUNIX OSのような機能規模が大きい基盤ソフトウェアが進出できるようになると考えます。
MetasepiとAjhcは、C言語より安全な設計を提供する次世代のLinuxとGCCを作る挑戦と言いかえることもできます。