データ構造に関する仕様を含め検証できるC言語プログラム部品検証ツール CForge

CForge Modular Verification Tool for C Programs Including Specifications of Data Structures

酒井　政裕　　今井　健男　　片岡　欣夫
■ SAKAI Masahiro　■ IMAI Takeo　■ KATAOKA Yoshio

東芝は，“仕様を基点としたソフトウェア高信頼化”というコンセプトで，ソフトウェアの信頼性向上に取り組んでいる。その一環として，仕様とプログラム部品を対応付けながらメンテナンスを行うことで，プログラムの改変・拡張時の信頼性を確保することを目指している。

今回，仕様とC言語プログラム部品の整合性の静的検証を行うツールCForgeを開発した。これはデータ構造とポインタを扱うことができ，また，仕様に基づいて網羅的に検証できる。

Toshiba has been engaged in a software reliability project based on the concept of "specification-leveraged dependability enhancement." The basic objective of this concept is to preserve consistency between a program and its specifications in the course of further modification or extension of the software.

We have developed CForge, a new modular verification tool that can verify consistency between C functions and their specifications. In addition, it is capable of comprehensive verification including data structures and pointers.

1　まえがき

ソフトウェアの不具合の多くは上流工程で発生し，そのほとんどが下流のテスト工程で発見されている。例えば組込みソフトウェアの分野では，ソフトウェア要求定義以前の工程で37％の不具合が混入しているのに対し，90％は単体テスト以降にならないと発見されないというデータがある（**図1**）。そのため，上流の不具合での後戻りコストが大きくなってしまっている。

また，現在のソフトウェア開発では従来製品の他機種への展開やシリーズ展開が多く，一度書かれたソフトウェアが幾度も拡張されたり再利用されることは一般的である。その際の書換えや環境の変化によって，ソフトウェアの信頼性が低下する“ソフトウェアの経年劣化（Software Aging）”[1]が問題となってきている。

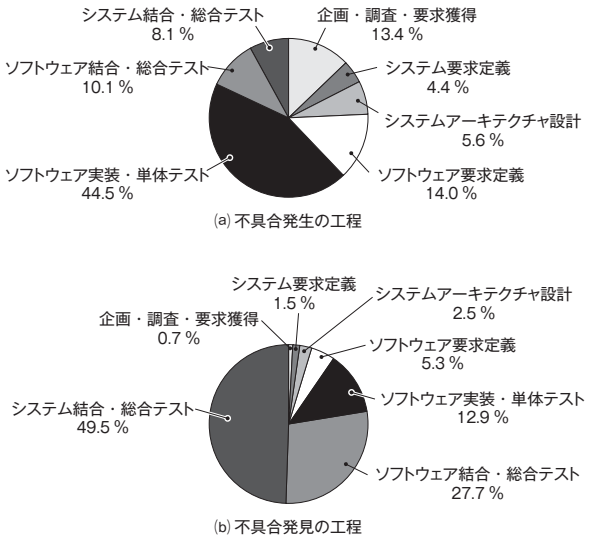
前者の問題を解決するためには，要求定義や設計の上流工程で品質を向上させ後戻りを削減することが有効であり，後者の問題を解決するためには，仕様とプログラムの対応をきちんと保証しながらメンテナンスを行うことが有効であると考えられる。

そこで東芝は，“仕様を基点としたソフトウェア高信頼化”というコンセプトに基づいた取組みを行っている。これは，要求定義から，設計，開発，保守，拡張，再利用までのソフトウェア ライフサイクルの中で，以下の二つの活動を継続的に行うことによって，仕様とプログラム相互の正しさを保証し，高信頼化を実現しようというものである（**図2**）。

（1）　ソフトウェア自体や対象ドメインに暗黙的に存在する制約を仕様に反映させ，また，環境の変化なども仕様に反映させることで，仕様自体を充実させ高品質化すること

（2）　プログラムと仕様の整合性を検証することによって，ソフトウェアの高信頼化を実現すること

当社は，このコンセプトの実現の一環として，C言語のプロ



＊情報処理推進機構
「2008年度組込みソフトウェア産業実態調査報告書」[2]に基づき作成。

図1．ソフトウェアの不具合発生と発見の工程 ― ソフトウェアの不具合の多くは上流工程で発生する一方で，その多くが発見されるのは下流のテスト工程である。

Introduction phase and detection phase of software defects

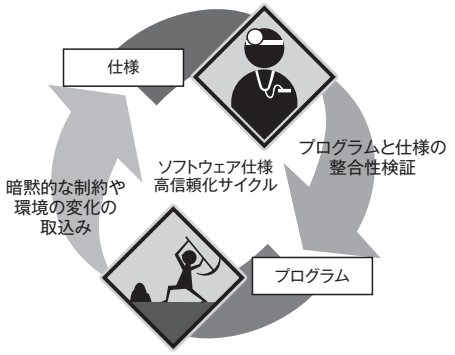


図2. ソフトウェア仕様の高品質化サイクル ― 仕様とプログラム相互の正しさを保証する仕様高品質化サイクルである。
Cycle of software specification enhancement

グラム部品と仕様の整合性を検証するツール CForge[(3)]を開発した。ここでは，CForgeのシステムの概要と簡単な使用例を中心に述べ，今後の展望についても述べる。

2　システムの概要

CForgeは，C言語プログラム（関数）のソースコードと仕様記述のファイルを入力として受け取り，プログラムが仕様を満たすかどうかの検証を行う。検証の結果，仕様を満たさない動作シーケンス（処理の実行順序）を発見した場合には，それを仕様に対する反例として出力し，検証にパスしなかったことを報告する（**図3**）。そのような動作シーケンスを発見できなかった場合には，検証にパスしたことを報告する。

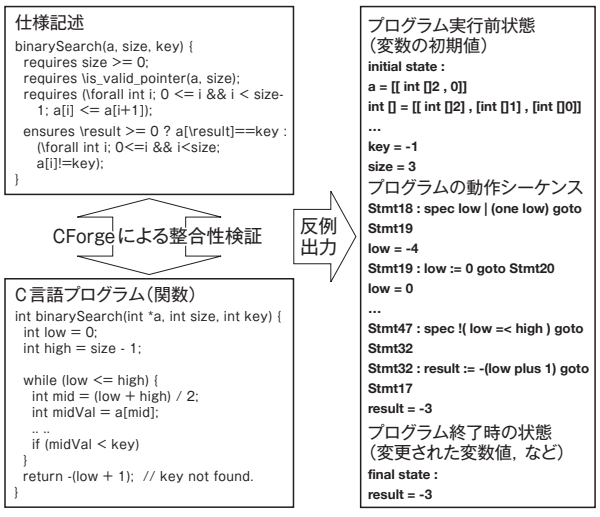


図3. CForgeの入出力 ― CForgeは仕様記述とC言語プログラム（関数）の整合性を検証し，検証の結果，仕様に違反する動作シーケンスを発見した場合には，仕様に対する反例として出力する。
Inputs and outputs of CForge

2.1　検証の特徴

CForgeによる検証は，実際にプログラムを動作させて行うテストなどの手法と異なり，プログラムを実際に実行せずに，ソースコードを解析することによって行う。このような手法は静的検証と呼ばれ，テストより早い開発工程で不具合を発見する手法として注目されている。

従来から利用されている静的検証ツールとしては，ソースコード検査ツールやモデル検査ツールが存在する。しかしCForgeによる検証は，これらのツールによる検証とは異なった特徴を持っており，相互に補完関係にあると言える。その特徴は以下のようなものである。

(1)　網羅的な検証　　ソースコード検査ツールの多くは，ヒューリスティックス[(注1)]に基づいてソースコードを検査することで典型的な欠陥を発見する。そのため，ヒューリスティックスから漏れてしまったものについては，簡単な欠陥であっても検出することができない。

それに対して，CForgeやモデル検査は想定されるあらゆる実行時のふるまい（動作シーケンス）のうち，指定された範囲内[(注2)]に収まるものを網羅的に検査することで検証を行う。そのため，この範囲内に仕様に違反する動作シーケンスがあれば確実に発見できる。

仕様に違反する動作シーケンスが指定した範囲外にしかない場合には，CForgeではそれを発見できない。しかし実際には，多くの欠陥は仕様違反を引き起こす比較的小さな実行シーケンスを持つという，小スコープ仮説（Small Scope Hypothesis）と呼ばれる経験的事実が知られており，このような検証は有効であると考えられている。

(2)　仕様に基づいた検証　　ソースコード検査ツールによる検査は，不正なメモリアクセスやデッドロックといった，典型的でかつ個別の仕様によらず判断できる誤りの発見に限られている。それに対して，CForgeはプログラム部品が仕様に適合しているかの検証を行う。そのため，個別の仕様によって判断できるような誤りを発見することができ，したがって個々のソフトウェアに固有の制約を検証することができる。

(3)　データ構造に関する検証　　一般的なモデル検査ツールは，プログラムの制御の流れに関する性質，例えば発生してはいけないイベントが決して発生しないことなどを検証するために作られている。そのため，プログラムの制御に関する性質の検証には優れているが，データ構造に関する複雑な条件などを検証することは難しい。それに対してCForgeは，データ構造に関する性質，例えば配列

(注1)　理論や原理に厳密には基づかないで解を得る方法で，必ずしも正しい解を得られるとは限らない。
(注2)　範囲内とは，CForgeの場合，実行に必要となるループの回数や使用メモリ量などで決まるサイズが一定以下であること。

がソートされているかなどを表現し，検証することが可能である。

2.2 仕様記述言語

CForgeは，前述のようにプログラム部品が仕様に適合しているかの検証を行うが，C言語には標準的な仕様記述言語が存在しないので，独自に設計した仕様記述言語を用いて仕様を記述する。この仕様記述言語の基本的な概念と構文は，Java(注3)言語の仕様記述言語として標準的に用いられているJML（Java Modeling Language）のものを採用し，C言語のポインタに関する性質，例えば有効なポインタであるかどうかなどを表現するための特別な構文を追加した。

CForgeの仕様記述言語は，C言語の関数の仕様を事前条件及び事後条件の形で記述する。事前条件は，関数の呼出し側が関数呼出し時に保証しなくてはならない条件である。事後条件は，事前条件が満たされた場合の関数の終了時に，関数が呼出し側に対して保証する条件である。このような形で仕様を記述することで，部品ごとの入出力仕様を明確にすることができるため，部品単位での検証が可能になる。

この仕様記述言語で記述した仕様の例を**図4**に示す。これは整数nの平方根を求めるC言語の関数**int** sqrt（**int** n）の仕様を記述したものである。**requires**（条件）;が事前条件を表しており，ここでは，引数nが非負であることを指定している。同様に**ensures**（条件）;が事後条件を表しており，ここでは，\result（関数の返り値）の2乗がn以下でかつ\result+1の2乗がnより大きいことを指定している。

```
sqrt(n) {
  requires n >= 0;
  ensures \result * \result <= n;
  ensures n < (\result+1) * (\result+1);
}
```

図4. 仕様の記述例 — 整数の平方根を求める関数の仕様記述例である。ただし，端数は切捨てである。
Example of specification

2.3 検証の仕組み

CForgeの検証は，入力されたプログラムと仕様記述ファイルを，それぞれ関係論理（Relational Logic）(4)と呼ばれる論理体系に基づいた中間形式へと変換し，それらをマサチューセッツ工科大学（MIT）で開発されているプログラム検証エンジンForge(5)を用いて検証することにより実現している。

関係論理は，データ構造の内部に存在するデータの参照関係を数学的に表現でき，また，データ構造に対する操作やデータ構造の性質も数学的に表現することができる。

（注3） Javaは，米国Sun Microsystems, Inc.の米国及びその他の国における商標又は登録商標。

ただし，Forge自体は，Javaを主な対象として開発されているため，C言語に存在するポインタの加減算などの演算を表現する直接的な方法を持っていない。そこで当社は，C言語のポインタ演算を，Fat Pointer(注4)(6)と呼ばれる手法を応用することで，関係演算で取り扱えるように対応付けた。

CForgeは現在，浮動小数点数，ポインタのキャスト（型変換），及び関数ポインタなどを除いた，C言語の基本的なサブセットで書かれたプログラムの検証を行うことができる。

3 検証例

CForgeを用いたプログラム部品の簡単な検証例として，二分探索アルゴリズムの実装の欠陥(7)を発見する例を述べる。この欠陥は整数オーバフローに関するもので，整数型の上限に近いサイズの配列を扱わないかぎり顕在化しないため，長い間有名なアルゴリズムの教科書である「Programming Pearls」(8)のサンプルコードやJava言語の標準ライブラリのコードに存在し続けていたものである。そのような欠陥もCForgeで検出することができた。

欠陥のある二分探索アルゴリズムをC言語で記述した例を**図5**に示す。また，この二分探索関数に期待される仕様を，CForgeの仕様記述言語で記述した例を**図6**に示す。ここでは，事前条件として，次の三つの条件を指定している。

（1） 引数sizeが非負である
（2） aの指す先にsize個の要素が存在する

```
int binarySearch(int *a, int size, int key) {
  int low = 0;
  int high = size - 1;
  while (low <= high) {
    int mid = (low + high) / 2;
    int midVal = a[mid];
    if (midVal < key)
      low = mid + 1;
    else if (midVal > key)
      high = mid - 1;
    else
      return mid; // key found
  }
  return -(low + 1);  // key not found
}
```

＊Bloch, J. Extra, Extra - Read All About It: Nearly All Binary Searches and Mergesorts are Broken(7)を元に作成。

図5. 欠陥のある二分探索関数の例 — low+highが整数の上限を超える場合に，整数オーバフローによりmidが負となり，a[mid]で範囲外アクセスをしてしまう。
Binary search function with one bug

（注4） Fat Pointerは，メモリブロックへの参照とメモリブロック中のインデックスの組でポインタを表現する手法であり，実行時にポインタのチェックを行うことでC言語プログラムを安全に実行するために提案された手法。

```
binarySearch(a, size, key) {
  requires size >= 0;
  requires \is_valid_pointer(a, size);
  requires (\forall int i; 0 <= i && i < size-1;
                a[i] <= a[i+1]);
  ensures \result >= 0 ? a[\result]==key :
                (\forall int i; 0<=i && i<size; a[i]!=key);
}
```

図6. 二分探索関数の仕様 — aがソートされた配列を指すポインタであることを事前条件とし，配列内にkeyが存在するならば返り値がその位置を表すことを事後条件としている。
Specification of binary search function

(3) aの指す先size個の要素は昇順に並んでいる

ここで \is_valid_pointer(a, n)は，aがn個以上の要素を持つ領域を指していることを表す構文であり，\forallは全称量化子である。

事後条件としては，返り値\resultが非負である場合にはa[\result]がkeyに等しく，\resultが負である場合にはaの指す先にkeyが存在しない，ということを指定している。

このコードと仕様をCForgeに入力し検証を行うが，動作シーケンスを探索する範囲として，整数のビット幅や，使用可能な領域のサイズ，ループ及び再帰の実行回数の上限などのパラメータを指定する必要がある。ここでは，整数はビット幅を3ビット，すなわち-4から3までの間で，使用可能な領域のサイズは各構造体型及び各配列型について三つ，ループ及び再帰の実行回数の上限は3回として，CForgeによる検証を行うものとする。

CForgeで検証を行うと1秒程度で終了し(注5)，検証結果では，keyがaの指す先に存在するにもかかわらず，負の値を返す動作シーケンスが仕様に対する反例として報告される。

この動作シーケンスを追跡すると，a[mid] で，midが負のため範囲外の領域を参照してしまい，それが仕様違反を引き起こしていることが確認できる。また，midが負になっている原因が，mid=(low+high)/2という代入の際に，low+highが整数の上限をオーバフローし，負の値になってしまっていることにあることが確認できる。

4 あとがき

C言語プログラム部品の検証ツール CForgeは，プログラム検証エンジンForgeを用い，C言語で記述された関数が仕様を充足するかの検証を行うツールである。

(注5) 検証にかかった時間は，Pentium® Dプロセッサー945 3.40 GHz，メインメモリ2Gバイトのマシンで1秒程度。Pentiumは，米国又はその他の国における米国Intel Corporation又は子会社の登録商標又は商標。

このようなツールを用いた検証を行いながら，開発を行うことで，仕様とプログラム部品の対応を保ちながら開発を進めることができる。また，仕様とプログラム部品を対応付けながら対でメンテナンスすることにより，特にシステムの改変・拡張時に，プログラム部品と仕様の対応が取れなくなることを防止することができる。その結果，システムの改変・拡張時の信頼性を保証し，ソフトウェアの経年劣化を防止できる。

現在のCForgeは，C言語の標準関数やアルゴリズムの初等的教科書に載っているアルゴリズムなどについて，実用的な時間で検証を行えることと，プログラムが仕様に合致していない場合に反例となる動作シーケンスを発見できることが確認できている。

今後は，CForgeを実用規模のソフトウェアに適用するため，C言語でまだ対応できていない機能に対して開発を進め，また，プログラムの規模に対する検証能力のスケーラビリティやユーザビリティの改善などを行っていく。

文 献

(1) Parnas, D. "Software Aging". Proceedings of the 16th International Conference on Software Engineering. Sorrento, Italy, 1994-05, ACM and IEEE Computer Society. p.279－287.
(2) 経済産業省；情報処理推進機構．“2008年度組込みソフトウェア産業実態調査報告書”．情報処理推進機構 ソフトウェアエンジニアリング．<http://sec.ipa.go.jp/reports/20080715.html>，(参照2009-04-15)．
(3) 酒井政裕，ほか．“CForge：C言語プログラムのための有界検査ツール”．第11回プログラミングおよびプログラミング言語ワークショップ論文集．岐阜，2009-03，日本ソフトウェア科学会 プログラミング論研究会．p.118－132.
(4) Jackson, D. Software Abstractions: Logic, Language, And Analysis. Cambridge, MA, The MIT Press, 2006, 366p.
(5) Dennis, G; Yessenov, K. Forge: Bounded Program Verification. <http://sdg.csail.mit.edu/forge/>，(参照 2009-04-15)．
(6) 大岩 寛，ほか．安全性を保証するANSI-C 実行系の実装手法．コンピュータソフトウェア．19，3，2002，p.195－200.
(7) Bloch, J. Official Google Research Blog: Extra, Extra - Read All About It: Nearly All Binary Searches and Mergesorts are Broken. <http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html>，(参照 2009-04-15)．
(8) Bentley, J. Programming Pearls. New York, Addison-Wesley, 1986, 208p.

酒井 政裕 SAKAI Masahiro
研究開発センター システム技術ラボラトリー。
ソフトウェアの設計・検証技術の研究に従事。
System Engineering Lab.

今井 健男 IMAI Takeo
研究開発センター システム技術ラボラトリー研究主務。
ソフトウェアの設計・検証技術の研究に従事。情報処理学会，日本ソフトウェア科学会会員。
System Engineering Lab.

片岡 欣夫 KATAOKA Yoshio, Ph.D.
研究開発センター システム技術ラボラトリー主任研究員，博士(情報科学)。ソフトウェアの生産性・品質向上技術の研究に従事。情報処理学会会員。
System Engineering Lab.