オブジェクトコード検証(OCV)の重要性：概要

www.ldra.com

© LDRA Ltd. This document is property of LDRA Ltd. Its contents cannot be reproduced, disclosed or utilized without company approval.

LDRA

目次

はじめに 3

オブジェクトコード検証：ツール認定例 3

「as-if」ルール 4

追加テストの考案 5

100%のオブジェクトコードカバレッジを達成する 6

DO-178C とオブジェクトコード検証 7

機能安全という異質な世界 7

どの程度の OCV で十分なのか？ 8

ツール認定パック OCV 8

アプリケーションコードでの単体テスト OCV 9

完全なアプリケーションコード上の OCV 9

結論 10

引用文献 11

はじめに

機能安全、セキュリティ、コーディング標準の規格（IEC 61508 [1]、ISO 26262 [2]、IEC 62304 [3]、MISRA C [4]、MISRA C++ [5]、CWE [6] など）がお墨付きを与える検証や妥当性確認のプロセスにおいては、テスト対象のアプリケーションがどの程度実行されたかを示すことが非常に重要であるとされます。経験上、すべてのコードが正しく実行されることが示されていれば、現場で不具合に遭遇する確率はかなり低いといえます。しかし、C/C++他、どれであれ、高水準言語でのソースコードに焦点が当てられたものです。このようなアプローチでは、コンパイラが開発者の意図を忠実に再現するオブジェクトコードを作成していることが前提になっています。

オブジェクトコードにおける制御フローとデータフローが、それの生成元であるソースコードの正確な鏡像になっていないことは必然ですので、ソースコードですべてのパスが確実に実行できることを証明しても、オブジェクトコードでも同じだと証明することにはなりません。さらに悪いことに、ソースコードの単体テストは、誰もが価値を認めるものですが、それも誤解を与える可能性があります。なぜなら、単体テストハーネスにラップされた関数をコンパイルして得られるオブジェクトコードは、完全なシステムのコンテキストで生成されたものとは著しく異なっているかもしれないからです。

航空宇宙産業で使用されている DO-178C [7] だけが、開発者の意図と実行コードの動作との間に危険な不整合が生じる可能性に着目している規格です。それでも、クリティカルアプリケーションすべてにおいて、ソースコードとオブジェクトコードの違いが壊滅的な結果をもたらし得るという事実に変わりはありません。

オブジェクトコード検証：ツール認定例

コンパイラが生成したオブジェクトコードの制御フロー構造が、元となるアプリケーションのソースコードとどのように異なるか、なぜ異なるかを理解するために、非常に単純な C ソースコード（LDRA tool qualification pack [8] からの引用）を見てみます。

```
void f_while4( int f_while4_input1, int f_while4_input2 )
{
  int f_while4_local1, f_while4_local2;

  f_while4_local1 = f_while4_input1;
  f_while4_local2 = f_while4_input2;

  while ( f_while4_local1 < 1 || f_while4_local2 > 1 )
    {
      f_while4_local1++;
      f_while4_local2--;
    }
}
```

次の関数呼び出し 1 回で、この C コードの 100% のソースコードカバレッジを達成できることがわかります。

```
f_while4(0, 3)
```

コードは、1 行 1 操作にリフォーマットして、フローグラフ上で、直線状のコード系列である「基本ブロック」をノードとして表現され(図 1)、ノード間の「有向エッジ」(線) で基本ブロック間の関係が表現されます。

「as-if」ルール

ソースコードをコンパイルすると、結果として得られるアセンブラコードのフローグラフは、ソースコードのフローグラフとはまったく異なるものになりがちです。C や C++コンパイラが従うルールでは、バイナリが*同じものであるかのように (as if)* 動作するのであれば、好きなようにコードを変更することが許されているからです。

C++規格 §1.9/1 [9]から

「*この規定は、as-if ルールと呼ばれることもある。なぜなら、実装では、観察可能なプログラムの振る舞いから判断できる範囲において、結果があたかもその要件に従っているかのようである限り、この文書のいかなる要件も無視してかまわないからである。たとえば、その値が使用されずプログラムの観察可能な動作に影響を与える副作用が発生しないと演繹できる場合、式の一部分を評価する必要はない*」

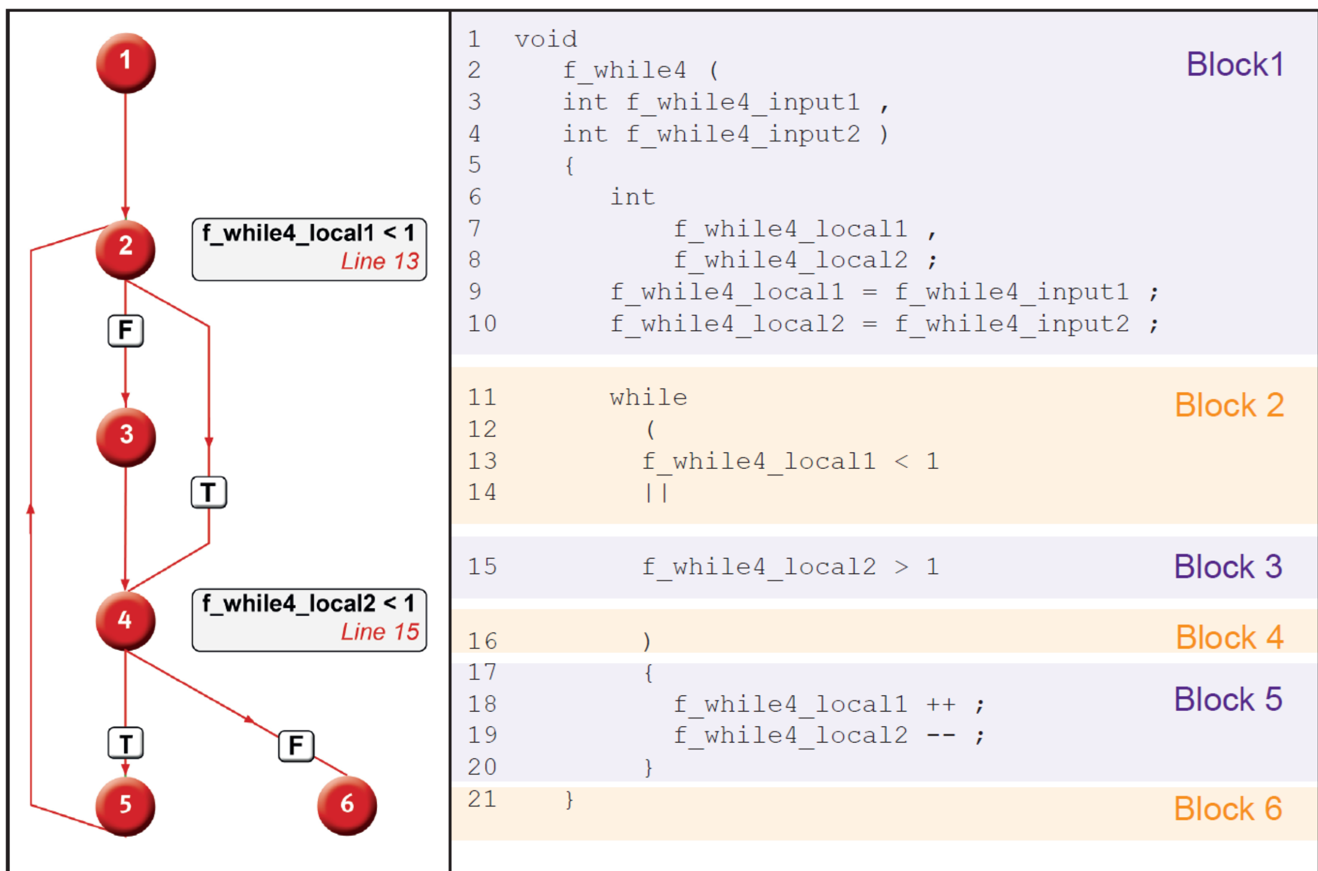

図 1:1 行 1 操作にリフォーマットし、フローグラフ上で
直線状のコード系列である「基本ブロック」をノードとして表現した元のソースコード

このコードを広く使用されている市販コンパイラでコンパイルすると、どれもたいていの場合、フローグラフは上記のソースコードに対するものとは異なるようになります(図 2)。図はこの例題で、TI 社コンパイラを最適化なしで使用した場合の結果を示すものです。フローグラフの赤色の要素は、以下の呼び出しによっては実行されていないコードを表しています。

```
f_while4(0, 3)
```

オブジェクトコードとアセンブラコードは 1 対 1 に対応するので、アセンブラコードを利用してオブジェクトコードでどの部分が未実行であるかを明らかにでき、テスト担当者が追加テストを考案して完全なアセンブラコードカバレッジを達成することで、オブジェクトコードカバレッジを達成させます。

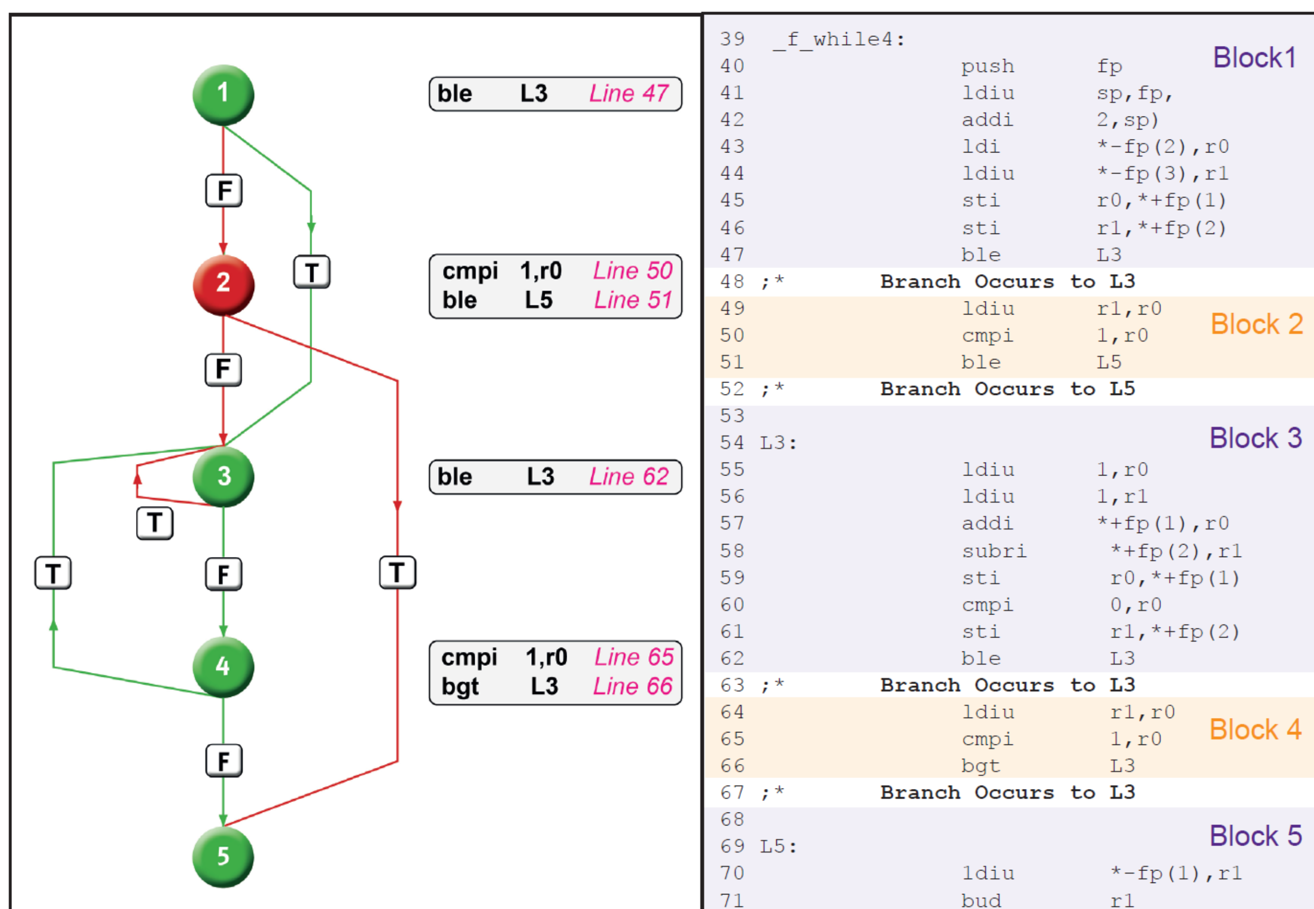

図 2：ソースコードでは 100%のカバレッジであるが、オブジェクトコードでのコードカバレッジ解析により、いくつか未実行のパスがあることが判明する

追加テストの考案

オブジェクトコードのロジックを調べることで、ソースコードレベルで追加のテストを考案することができます。たとえば、62 行目(Block 3 の最後)の `ble` 条件分岐命令(L3 への分岐)は、現時点のテストデータでは Block 3 のループは 1 回しか繰り返されないため、条件が false と評価され、繰り返しを続行するかどうかを確認するテストの 2 つの可能な結果の 1 つだけが発生しています。ソースコードで以下の新しいテストケースを追加し、true のケースが実行されてループが 2 回目の繰り返しを実行するようにします(図 3)。

```
f_while4(-1, 3)
```

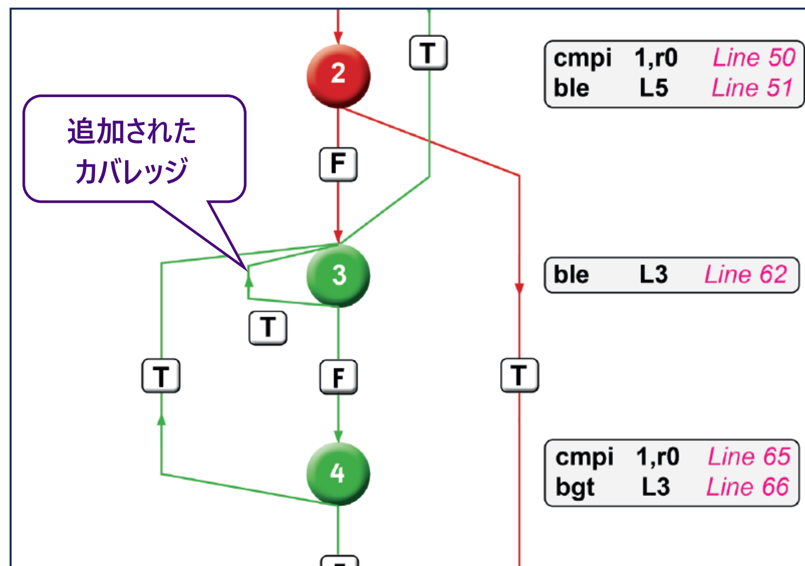

図3：テストケース f_while4(-1, 3) を適用した場合に得られる追加のオブジェクトコードカバレッジ

100%のオブジェクトコードカバレッジを達成する

同様のアプローチで 47 行目と 52 行目で分岐のオブジェクトコードのロジックを調べることで、さらにテストケースが 2 つ作成されます。したがって、この例のすべてのオブジェクトコードを実行する「main」関数は、次のようになります。

```
int main()
{
  int a = 0;
  int b = 1;
  int c = 0;

  a = b || c;

  /* WHILE_WITH_OR.C */

  f_while4(0, 3); // Achieves 100% source code coverage
  f_while4(-1, 3);
  f_while4(3, 3);
  f_while4(3, 0);

  return(0);
}
```

LDRA

DO-178Cとオブジェクトコード検証

上記の前提を踏まえて、DO-178Cと航空宇宙領域で要求されるもの、特に「構造カバレッジ解析」と題された 6.4.4.2 項に戻ります。DO-178Cが構造カバレッジを要求する理由は、以下のように、構造カバレッジが、要件ベーステストでアプリケーションを完全に実行したことを保証する最良の方法だからです。

「*この解析の目的は、要件ベースのテスト手順でどのコード構造が実行されなかったかを判断することである。要件ベースのテストケースではコード構造が完全には実行されていない可能性があるため、構造カバレッジ解析を実施して、構造カバレッジを得るための追加の検証を生成する*」

パラグラフ 6.4.4.2b では、次のようにアプリケーションがレベル A である場合の追加の要件について説明しています。

「*構造カバレッジ解析は、ソフトウェアがレベル A ではないか、コンパイラがソースコードの文に直接トレースできないオブジェクトコードは生成しないという場合には、ソースコードに対して実施してもよい。その場合、生成されたオブジェクトコード系列の正しさを証明するために、オブジェクトコードに対して追加の検証を行う必要がある。…*」

つまり、アプリケーションのレベルがAであって、コンパイラがソースコードにトレースできないオブジェクトコードを生成している場合には、オブジェクトコードの追加検証を行う必要があるのです。しかし、DO-178Cでは、この作業を自動化する義務もオブジェクトコードのカバレッジを達成する義務もまったくありません。この点を強調するため、2002年のFAAのポジションペーパーCAST-12「Guidelines for Approving Source Code to Object Code Traceability」[10]（現在FAAのサイトにリンクなし）では、手作業によるレビューをどのように行うかだけでなく、そのような手作業による解析に費やす時間を最適にするために、典型的な構造を使用してサンプルアプリケーションを考案する方法についても概説されています。

機能安全という異質な世界

このようなアプローチの問題点は、コンパイラ開発者が満たすべき仕様と、「as-if」原則に基づいてコンパイラ開発者に与えられた柔軟性にあります。機能安全なコードには、さまざまな原因で起こり得る予期せぬ事象から守るための防御的コードを含める必要があります。たとえば、コーディングエラーや宇宙線によるメモリの破損は、コードのロジックからは「不可能な」コードパスが実行されることにつながる可能性があります。

高水準言語、特にC/C++にはコードが準拠する言語仕様で動作が規定されていない機能が驚くほど多くあります。定義されていない動作が、取り除くことができず潜在的に悲惨な結果につながることは明らかであり、これは機能安全なアプリケーションにおいて防御されなければなりません。コードには高いレベルのコードカバレッジを実現することも要求され、一部の（特に自動車）分野では、高度な外部診断や、キャリブレーション、そして開発用のツールに対応した設計が要求されることが非常に一般的です。

LDRA

しかし、防御的コーディングや外部データアクセスなどのプラクティスは、コンパイラが認識する世界に属するものではないということです。たとえば、C も C++ もメモリ破壊を考慮していません。そのため、メモリ破壊から保護するように設計されたコードは、メモリ破壊がないときにアクセスできなければ、最適化で無視されてしまう可能性があります。したがって、防御的コードは、それが「最適化で削除」されないために、構文的にも意味的にも到達可能でなければなりません。

未定義の動作も予期せぬ事態の原因になり得ます。単に避けるべきだと言うのは簡単ですが、多くの場合、それらを特定することが非常に困難です。未定義の動作が存在する場合、コンパイルされた実行コードの動作が開発者の意図と一致する保証はありません。また、デバッグツールで使用されるデータへの「バックドア」アクセスも言語が考慮していない別の状況を表しているため、予期しない結果をもたらす可能性があります

これらすべての領域はコンパイラベンダーの検討事項には含まれていないため、コンパイラの最適化が大きな影響を与える可能性があります。一見健全な防御的コードが「実行不可能性」に結びつく場合、つまり、どのような入力値の集合によってもテスト・検証できないパス上に存在する場合に、それが最適化によって削除されてしまう可能性があります。さらに憂慮すべきは、単体テスト中に存在すると示された防御的コードが、システム実行ファイルが構築されるときに削除される可能性があることです。つまり、単体テスト中に防御的コードのカバレッジが達成されたからといって、完成したシステムにその防御的コードが存在することは保証されないのです。この問題の潜在的な影響を示す詳細な例は、LDRA のホワイトペーパー「オブジェクトコード検証：なぜ重要なのか」[11]にあります。

どの程度の OCV で十分なのか？

DO-178C の要件に注目すると、DO-178B に対して追加して明確化したにもかかわらず、この点に関して規格は贔屓目に見ても曖昧なままです。開発チームには、行っている作業と、その作業によってエラーが発生した場合のリスクを考慮した上で、正当化できる方法を選ぶ責任があります。

以下で説明する 3 つの手法において、オブジェクトレベルで達成された異なるカバレッジ結果の例は、コード構造が違っていることを示すものであるため、その違いを確認して文書化する機会を提供します。強調されるべきことは、オブジェクトコードに追加された（ソースにはない）分岐構造が実行不可能であることは非常に多く、したがってオブジェクトコードレベルで100%のカバレッジを達成することは不可能だということです。それはコード実行の目的ではないため、できなかったからといって、規格の DAL-A 要件を満たさないことを意味するものではありません。

ツール認定パック OCV

この例の関数 `f_while4` は、LDRA のツール認定サポートパックから引用したものです。このサポートパックは、セーフティクリティカルなシステムで使用される可能性のあるすべてを網羅する要素から構成されており、構成要素それぞれが個別に証明できるものです。これによって、コンパイラがすべての構成要素を正しく解釈できることが証明されていれば、それら構成要素が完全なアプリケーションの部分として使用される場合は信頼性が確保されているという考え方を裏付けることができます。LDRA ツールスイートは、このようなアプローチをサポートする機能を備えています（図 8)。

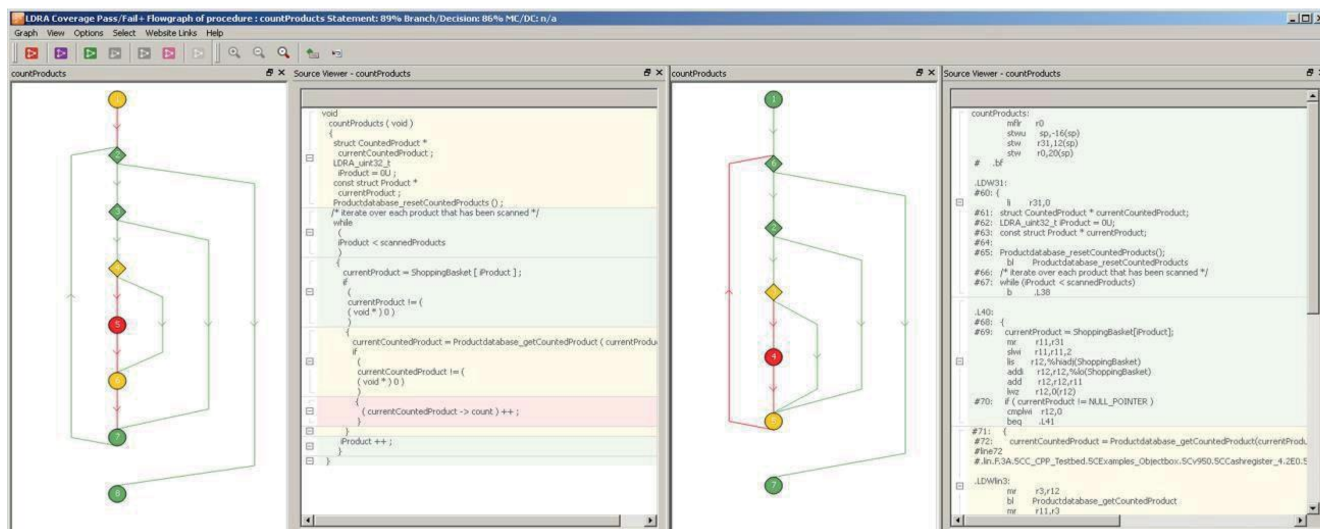

図 8：LDRA ツールスイートを使用したソースコードとオブジェクトコードのカバレッジの比較

アプリケーションコードでの単体テストOCV

より一般的な妥協案は、ソースコード解析を完了するために利用したものと同じ単体テストに OCV を適用してカバレッジ結果を比較し、オブジェクトコードとソースコードの間で異なっている結果があればそれを調査することです。

これがより良い解決策であることは明らかです。このシナリオでは、ツール認定パックの例とは違ってテスト対象のコードはアプリケーションの一部です。ある関数に対してテストハーネスの一部として生成されたオブジェクトコードと、同じ関数に対して完全なアプリケーションコードベースの一部として生成されたオブジェクトコードとの間の不一致の範囲は、「as-if」ルールの結果として残ります。しかし、アプリケーションコードは、DO-178C 規格で明確に規定される高レベルコードとオブジェクトレベルコードの構造の違いをすばやく簡単に特定する方法を提供します。

完全なアプリケーションコード上のOCV

とはいえ、「as-if」ルールの影響で悪化する、コンパイルされたコードと開発者の意図との間の不一致の範囲を無視することは容易ではありません。このことは、最も正確で包括的なテストは完全なアプリケーションの OCV を実施することによってのみ保証されるということを明確にしています。LDRA ツールスイートが提供する機能が、それを可能にします。

完全なアプリケーションコード上の OCV は必要でしょうか? もちろん、DO-178C 他どの規格から見ても必要ではありません。ベストプラクティスでしょうか? その立場に異議を唱えるのは難しいです。商業的に正当化されますか? それは開発チームが判断することです。

結論

本稿では高水準言語として C を中心に説明しますが、強調している問題は多かれ少なかれ、言語に関係なく当てはまります。たとえば、C++ 全般の性質、特に C++14 や C++17 など後のバージョンで導入された拡張機能は、実行ファイル（オブジェクトコード）内のより多くの要素がソースまで追跡できないことを意味するものです。

例題は非常にシンプルなものです。もちろん、実プロジェクトでは考慮すべきコードの規模はもっと大きく、多くは残りのコードベースと同レベルで厳密な解析を必要とするライブラリなどサードパーティのコードを含んでいます。このような場合のベストプラクティスは、認定されたライブラリにデプロイするか、サードパーティのコードを内部開発のコードと同じ妥当性確認および検証作業の対象にすることです。

本稿で論じた OCV の適用に対するさまざまなアプローチは、ソースコードとオブジェクトコード間の不整合を検査し、正当化するための全体的な戦略の一部として考えることが最適です。また、手作業によるコード検査や、実行パスをトレースするためのデバッグツールの使用など、他の手法と組み合わせて、システムのさまざまな部分に適用することができます。

DO-178C のレベル A（すなわち DAL A）のアプリケーションは、危険にさらされているクリティカルアプリケーションを代表するものです。オブジェクトコード検証（OCV）は、規格の要求が満たされていることを確認するうえで非常に有用なツールを提供します。しかし、これらの要求は曖昧であり、何が適切で必要かを決定することが開発チームの判断に委ねられているのは事実です。そして、それは究極のベストプラクティスではないかもしれませんが、現実的な妥協です。

OCV は、どのセクターのいかなる規格でも義務付けられていません。民間航空セクターにおいてさえ、CAST や他のアプローチを述べた論文があるように、OCV を回避する方法があります。しかし、OCV を完全に回避することは、本稿で説明するものよりもさらに妥協であり、ベストプラクティスから離れることを意味します。

コンパイラ開発者の検討事項と機能安全の要求との間の不一致に起因する問題は、OCV を使用しなければ検出されないかもしれません。高度にクリティカルなソフトウェアアプリケーションに使用されるコンパイラは一般的に、非常に稀な状況を除いて設計基準を満たしていると想定できます。しかし、その設計基準には機能安全への検討事項は含まれていません。オブジェクトコード検証は、現在のところ、それらの検討事項を信頼の輪の中に取り込むための最も確実なアプローチです。

引用文献

[1] International Electrotechnical Commission (IEC), IEC 61508:2010 "Functional safety of electrical/electronic/programmable electronic safety-related systems" (all parts), IEC, 2010.

[2] International Organization for Standardization, ISO 26262:2018 "Road vehicles - Functional safety", International Organization for Standardization, 2018.

[3] International Electrotechnical Commission, IEC 62304:2006+AMD1 Medical device software - Software life cycle processes, IEC, 2015.

[4] MISRA, "MISRA C:2012 Third Edition, First Revision," The MISRA Consortium Limited, February 2019. [Online]. Available: https://www.misra.org.uk/product/misra-c2012-third-edition-first-revision/. [Accessed 24 September 2021].

[5] MISRA, "MISRA C++:2008," The MISRA Consortium Limited, June 2008. [Online]. Available: https://www.misra.org.uk/product/misra-c2008/. [Accessed 24 September 2021].

[6] Homeland Security Systems Engineering and Development Institute, "CWE Common Weakness Enumeration," The MITRE corporation, August 2021. [Online]. Available: https://cwe.mitre.org/. [Accessed 24 September 2021].

[7] RTCC, DO-178C "Software Considerations in Airbourne Systems and Equipment Certification", RTCA, 2011.

[8] LDRA, "LDRA Tool Qualification Support Packs (TQSP)," 2021. [Online]. Available: https://ldra.com/product/tool-qualification-support-pack-tqsp/. [Accessed 24 September 2021]. (https://ldra.com/products/tool-qualification-support-packs-tqsp/)

[9] ISO, "ISO/IEC 14882:2020 Programming languages — C++," December 2020. [Online]. Available: https://www.iso.org/standard/79358.html. [Accessed 24 September 2021].

[10] Certification Authorities Software Team, "Position Paper CAST-12 - Guidelines for Approving Source Code to Object Code Traceability," CAST, 2002.

[11] LDRA, "Object Code Verification: Why it matters," LDRA, Monks Ferry, 2021.

www.ldra.com

LDRA

LDRA UK & Worldwide
Portside, Monks Ferry,
Wirral, CH41 5LH
Tel: +44 (0)151 649 9300
e-mail: info@ldra.com

LDRA Technology Inc.
2540 King Arthur Blvd, 3rd Floor, 12th Main Lewisville Texas 75056
Tel: +1 (855) 855 5372
e-mail: info@ldra.com

LDRA Technology Pvt. Ltd.
Unit B-3, Third floor Tower B, Golden Enclave
HAL Airport Road Bengaluru 560017
Tel: +91 80 4080 8707
e-mail: india@ldra.com

FUJI SETSUBI

富士設備工業株式会社 電子機器事業部
http://www.fuji-setsu.co.jp