

Polyspace ハンズオンセミナー ソフトウェアバグを根絶するC/C++静的コード解析

MathWorks Japan

アプリケーションエンジニアリング部

アプリケーションエンジニア

田中 康博

アジェンダ

第1部 静的コード解析手法を提供するPolyspace紹介

第2部 Polyspaceを利用した静的コード解析

2.1 Polyspaceプロジェクト作成

2.2 レビュー時のTips & 便利機能

2.3 まとめ

第3部 Q&A会

バグを見つけることができますか？

```
ソース
検索結果の統計 WhereAreTheErrors.c
1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default*/
7     tmp = 0; /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11     while (actuator_position < 10)
12     {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16     }
17     if ((3*magnitude + 100) > 43)
18     {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23     return actuator_position*magnitude; /* value */
24 }
```

ゼロ割の可能性

オーバーフローの可能性

未初期化の可能性

actuator_position = x / (x - y);

Polyspace のコード証明

```

1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default*/
7     tmp = 0; /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11     while (actuator_position < 10)
12     {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16     }
17     if ((3*magnitude + 100) > 43)
18     {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23     return actuator_position*magnitude;
24 }

```

✓ あらゆる条件でコードの安全性を証明

✓ テストを行わずに網羅的に解析

- **実行時エラー**
- **条件による実行時エラー**
- **到達不能なコード**

✓ ゼロ除算 ?

Scalar division by zero does not occur
operator / on type int 32

left: 10

right: [-21474855 .. -1]

result: [-10 .. 0]

actuator_position = $x / (x - y)$;

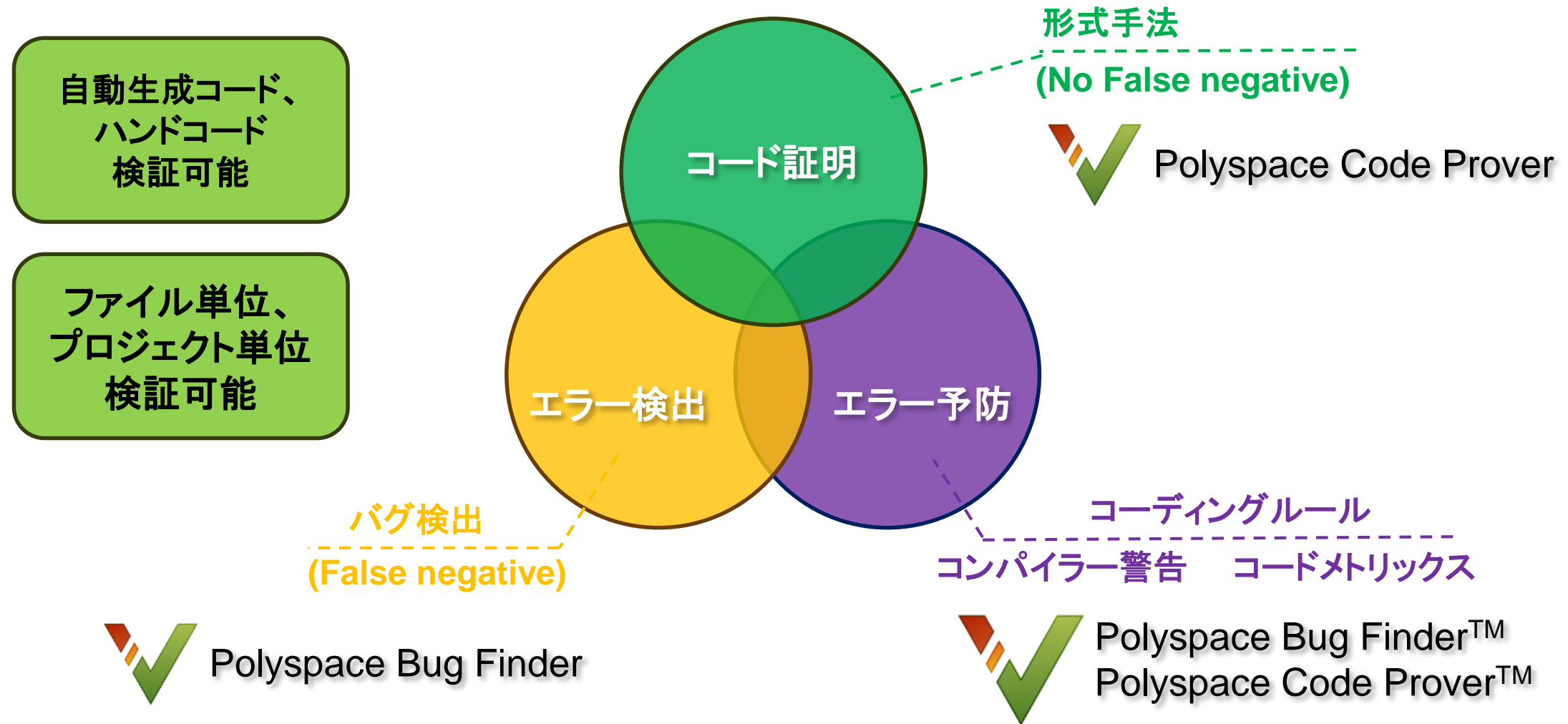
アジェンダ

Polyspaceの概要

- Polyspaceプロジェクト作成
- レビュー時のTips & 便利機能
- まとめ

Polyspaceの静的解析ソリューション

Polyspaceはコードの信頼性を確保するための機能を提供



Polyspace ユーザーリスト

- 航空宇宙・防衛
- 自動車
- 産業装置
- 交通・運輸
- 家電製品
- 医療機器



ランタイムエラーとは？

- プログラム実行中に発生するソフトウェアの不具合
- プログラムのクラッシュや暴走等をもたらす

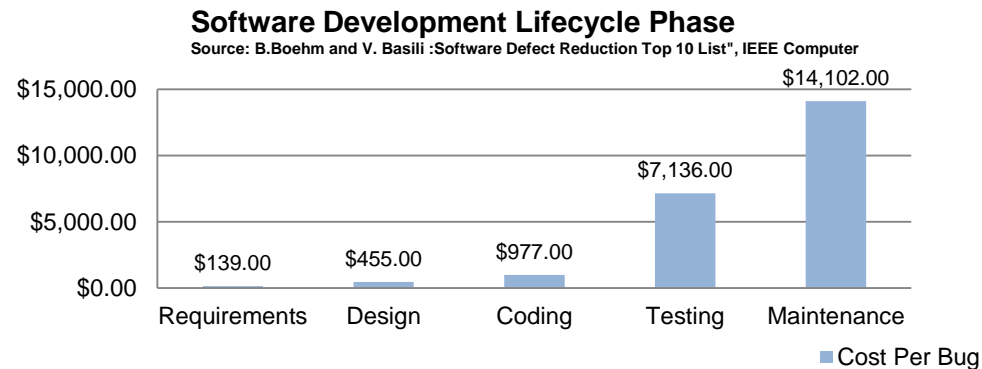
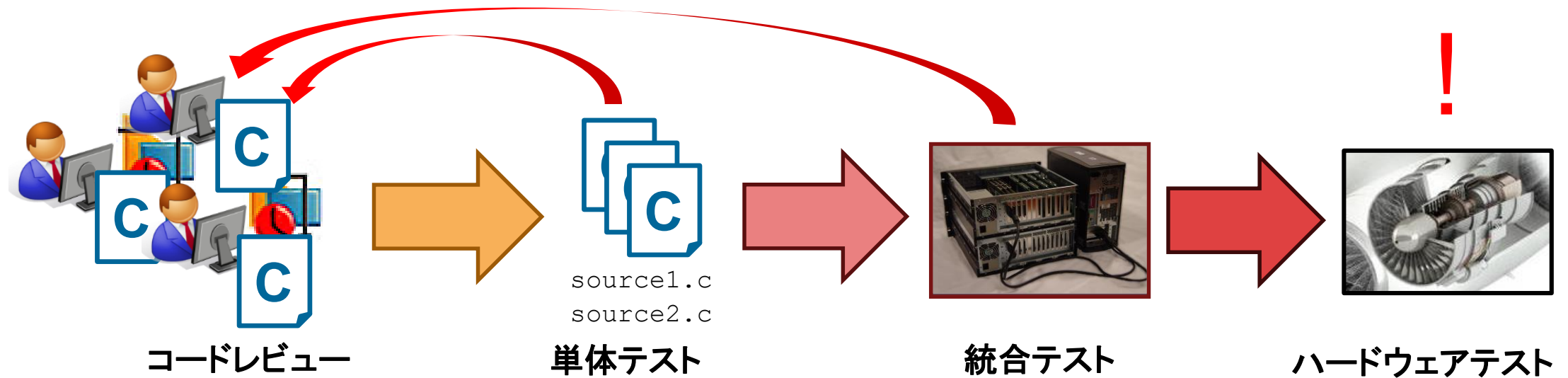
- 未初期化変数の参照
- 配列外アクセス
- ゼロ割
- 不正なポインタ参照
(NULL参照、領域不足)
- オーバーフロー・アンダーフロー

- 不正な型変換
(小さな型への変換による
オーバーフロー)
- 不正な数学関数コール
(負数の平方根等)
- デッドコード
- 設計標準規約※

Polyspaceはランタイムエラーの有無を証明する！

開発後期でのバグ検出の危険性

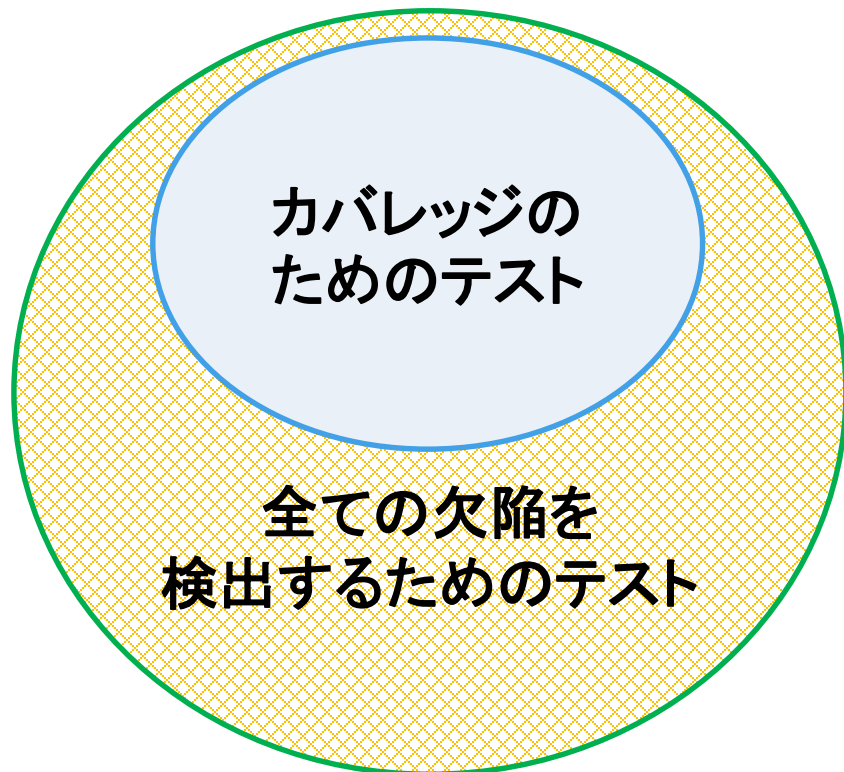
動的テスト完了後・製品リリース後にエラーが判明した！



ソフトウェア品質確保の効率化に向けて

提案：静的解析によるソフトウェア品質の確保

- 従来のテスト手法 → テストケースの作成、実行、確認が必要



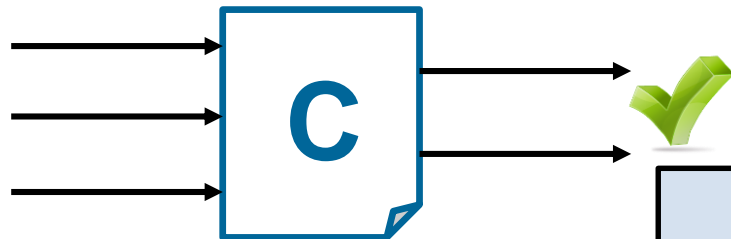
$$x / (x - y)$$

- 未初期化変数 `int32` の場合、
- オーバーフロー 4.61×10^{18} の
- ゼロ除算 テストケースが可能

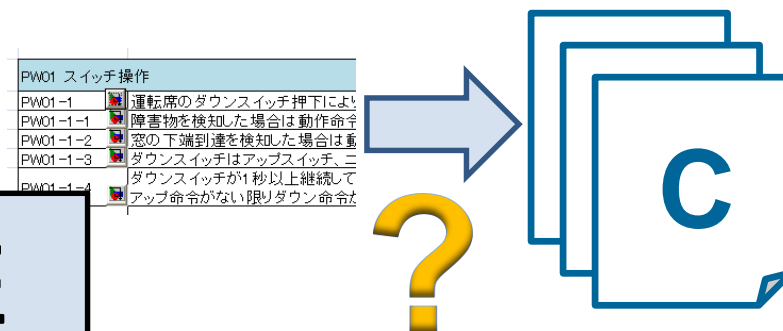
全て可能なテストケースの
作成・実行・レビューは不可能！

コード検証の目的

要求仕様通りの設計かを確認

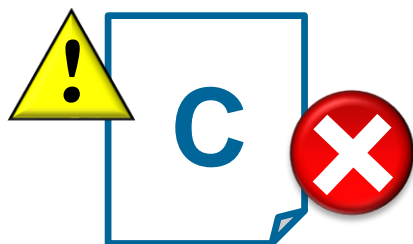


要求仕様に問題ないことを確認



動的検証

不具合(ゼロ割等)がないことを確認



静的検証

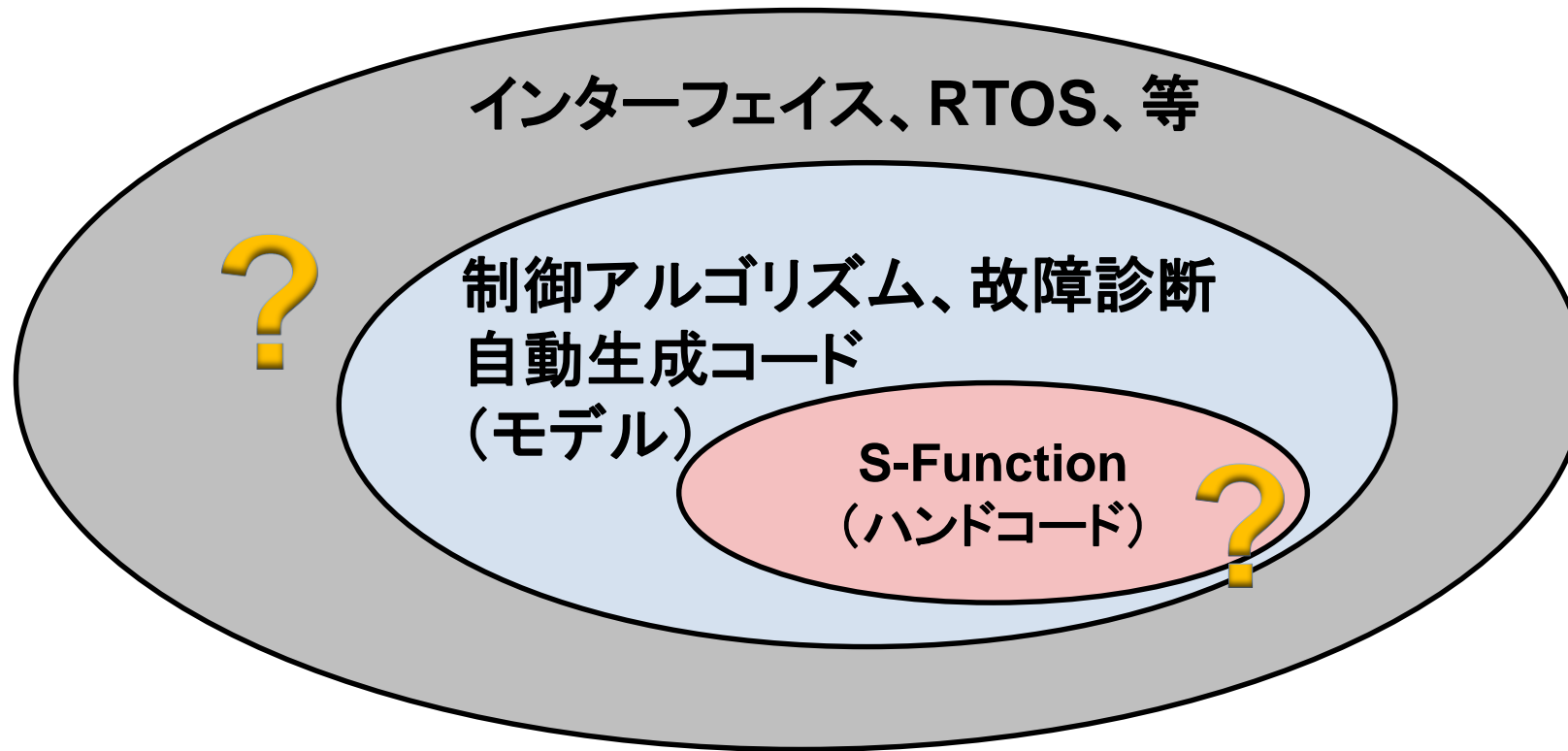
コーディングルール準拠の確認



適切な検証手法を使用してプロセスを効率化

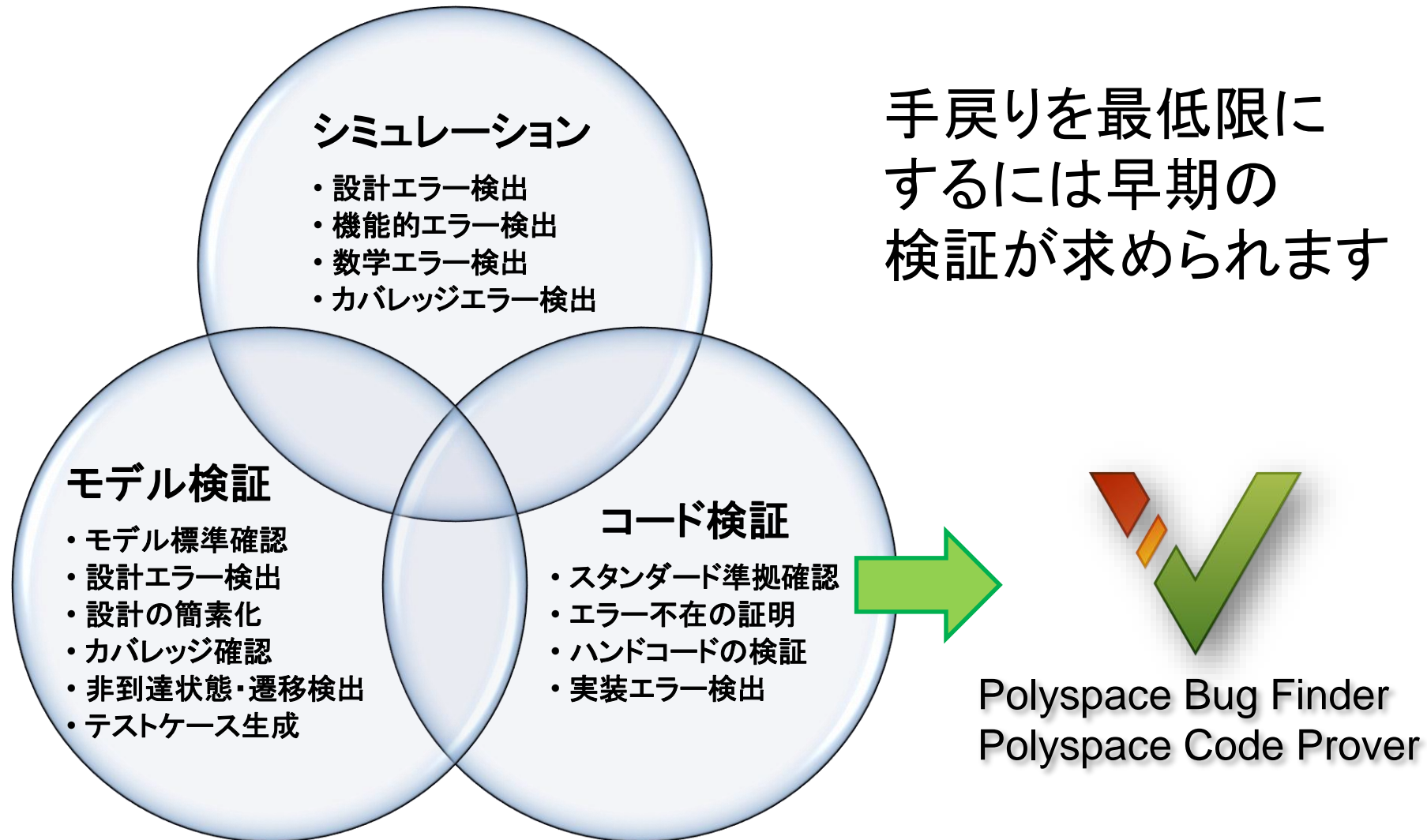
モデルベースデザインでも

自動生成コードをハンドコードと統合していませんか？



コード検証が必要になります

モデル検証とコード検証の役割分担

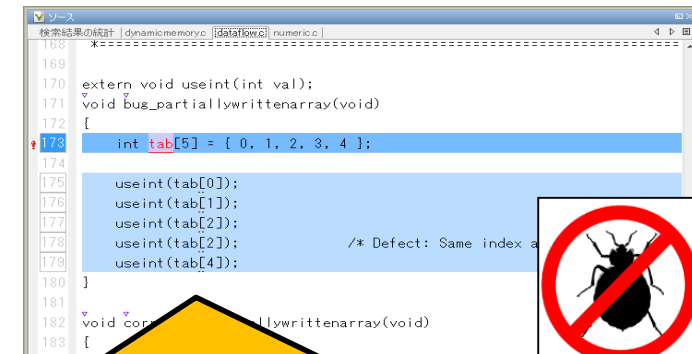




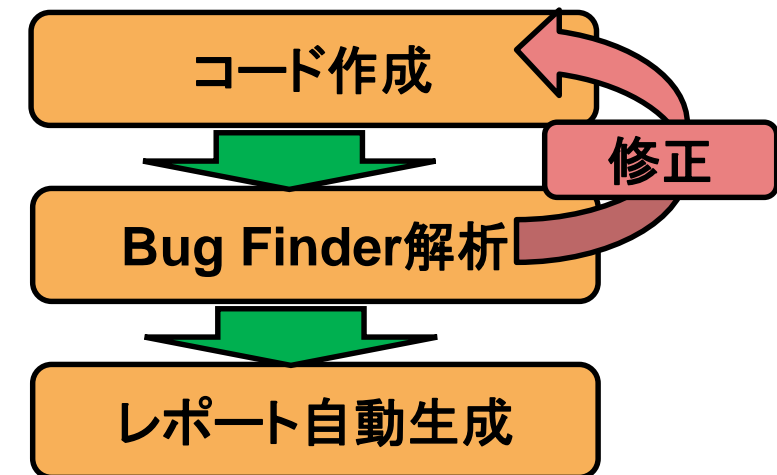
Polyspace Bug Finderのスピーディー解析

- セキュリティ脆弱性・バグ検出
 - コード作成後、すぐに欠陥を検出・修正
- コーディングルールチェック
 - エラー予防と再利用性向上
 - MISRA準拠を確認
- コードメトリックス解析
 - コードの複雑度を測定

開発プロセスの上流で不具合を発見！
コードを統合前に多くの欠陥を修正！
コード開発の効率に繋がる！



時間を掛けずに大部分のバグを識別



素早い欠陥検出・修正を可能とするPolyspace Bug Finder

セキュリティ脆弱性・バグ検出項目 (抜粋)

数値

- ✓ ゼロ割、オーバーフロー
- 負の値のシフト
- ...

動的メモリ

- メモリリーク
- ポインターの無効な解放
- ...

データフロー

- 読取りのない書き込み
- 部分的にアクセスされる配列
- ...

リソース管理

- 読取り専用リソースに書き込み
- 以前に閉じられたリソースを使用
- ...

静的メモリ

- ✓ 配列の範囲外アクセス
- NULLポインター
- ...

セキュリティ

- パス操作が脆弱
- 疑似乱数発生器が脆弱
- ...

同時実行

- ✓ データレース
- ✓ デッドロック
- ...

適切な手法

- 未使用のパラメータ
- 値渡しの大きな引数
- ...

プログラミング

- 配列の初期化が不適切
- 宣言の不一致
- ...

汚染されたデータ

- 汚染された文字列形式
- 汚染されたポインターの使用
- ...

オブジェクト指向

- "explicit"キーワードがない
- コンストラクターで未初期
- ...

暗号化

- 脆弱な暗号アルゴリズム
- 予測可能な暗号キー
- ...



サイバーセキュリティ – 業界活動と標準

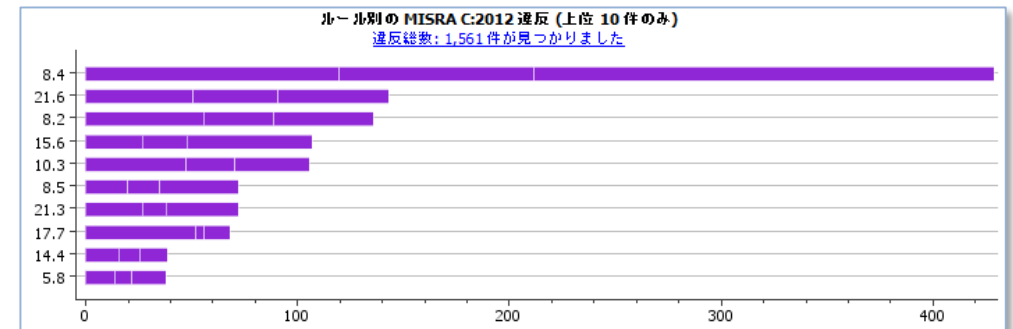
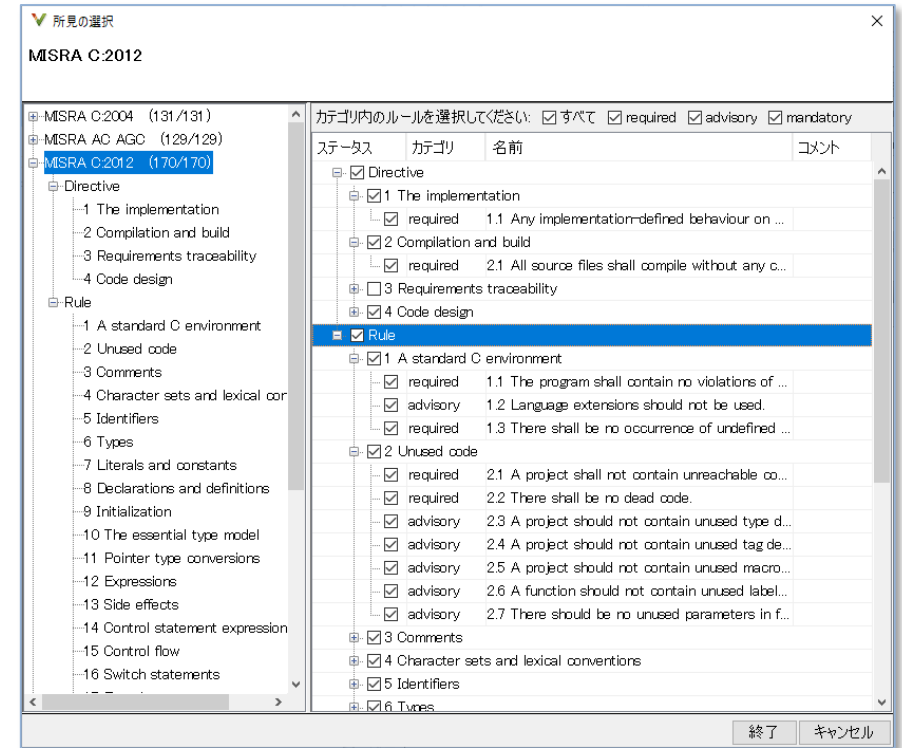
自動車業界に見られるコーディング標準 & 実践

- **CERT C** : セキュアコーディングスタンダード
- **ISO/IEC TS 17961** : Cセキュアコーディングルール
- **CWE** : 共通脆弱性タイプ
- **MISRA-C:2012** Amendment 1 : セキュリティガイドライン



コーディングルールチェック

- **MISRA C[®] : 2004** チェッカー
- **MISRA C[®] : 2004 AC AGC** チェッカー
 - 自動生成コード用
- **MISRA C[®] : 2012** チェッカー
- **MISRA[®] C++** チェッカー
- **JSF[®]++** チェッカー
- **AUTOSAR C++14** チェッカー



コードメトリクス

プロジェクト メトリクス (8)

- 再帰の数
- 直接再帰の数
- ファイルの数
- ヘッダー ファイルの数
- 最大スタック使用量のプログラミング *1
- 最小スタック使用量のプログラミング *1
- 保護されている共有変数の数 *1
- 保護されていない可能性のある共有変数の数 *1

ファイル メトリクス (4)

- コメント密度 *2
- 推定の関数結合
- コメントなし行の数
- 行数

関数 メトリクス (19)

- 呼び出し元関数の数
- 呼び出された関数の数
- 本体内の行数
- 実行可能行数
- GoTo ステートメントの数
- 呼び出しレベルの数
- ローカルの静的変数の数
- ローカルの非静的変数の数
- ローカル変数サイズのより高い推定値
- ローカル変数サイズのより低い推定値
- 最大スタック使用量 *1
- 最小スタック使用量 *1
- 呼び出しの発生数
- 関数パラメーターの数
- パスの数
- Return ステートメントの数
- 命令の数
- 循環的複雑度
- 言語スコープ

循環的複雑度

関数本体における線形独立パス数

[このページをすべて展開する](#)

説明

このメトリクスは関数内の判定点の数を計算し、合計に 1 を足します。判定点とはプログラムを 2 つのパスに分岐させるステートメントです。

このメトリクスの推奨上限は 10 です。循環的複雑度が高い場合、コードは可読性が低く、かつオレンジ チェックが増加する可能性があります。したがって、このメトリクスの値を制限するようにしてください。

メトリクスの制限を適用するには、以下のようにします。

- Polyspace® のユーザー インターフェイスで、コード複雑度メトリクスの計算を参照してください。
- Polyspace Metrics Web インターフェイスで、メトリクスとソフトウェア品質目標との比較を参照してください。

*1 : Code Proverで測定可能

*2 : しきい値は下限値

下線 : [HISコード複雑度メトリクス](#)

<https://jp.mathworks.com/help/bugfinder/metrics-reference.html>



Polyspace Code Proverでコードの正しさを証明

- **Quality (品質)**
 - ランタイムエラーの証明
 - 測定、向上、管理
- **Usage (使用方法)**
 - コンパイル、プログラム実行、テストケースは不要
 - 対応言語: C/C++/Ada
- **Process (プロセス)**
 - ランタイムエラーの早期検出
 - 自動生成コード、ハンドコードの解析可能
 - コードの信頼性を測定

実機実験前にコード信頼性を
確保して手戻りを削減

グリーン: 正常

ランタイムエラーが存在しない

レッド: エラー

実行される度にランタイムエラー

グレー: デッドコード

無実行

オレンジ: Unproven

条件によってランタイムエラー

パープル: Violation

MISRA-C/C++, JSF++

変数値範囲

ツールチップ

```
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;

    for (i = 0; i < 100; i++) {
        *p = 0;
        p++;
    }

    if (get_bus_status() > 0) {
        if (get_oil_pressure() > 0) {
            *p = 5;
        } else {
            i++;
        }
    }

    i = get_bus_status();

    if (i >= 0) {
        *(p - i) = 10;
    }
}
```

variable 'i' (int32): [0 .. 99]
assignment of 'i' (int32): [1 .. 100]

Polyspaceは全ての実行パスの結果を証明する！

Polyspace形式検証によるMISRAルール違反の正当化

MISRA
ルール違反

安全性の証明
(オーバーフロー無し)

The screenshot displays the Polyspace Code Prover interface. The main window shows a C code snippet with a MISRA C:2004 violation (ID 7) and an overflow warning (ID 130). The violation details are shown in a yellow box, and the overflow analysis is shown in a green box. The right-hand pane shows the 'Check Review' for the MISRA C:2004 10.1 rule, indicating that the violation is 'Not a defect' and 'Justified'.

```
13 float risk_of_floatdivisionbyzero(int p)
```

ID 7: MISRA C:2004 10.1
The value of an expression of integer type shall not be implicitly converted to a different underlying type.
Implicit conversion of the binary - right hand operand of underlying type 'signed int' to 'float' that is not an integer type. (Required)

ID 130: Overflow
Operation [-] on float does not overflow in FLOAT32 range
operator - on type float 32
left: 1.0
right: [-2.1475E+09 .. -0.9999] or [0.0 .. 2.1475E+09]
result: [-2.1475E+09 .. 2.1475E+09]

```
19 tmp = j - p;
```

```
25 i = 0.0;  
26 }  
27 return i;  
28 }
```

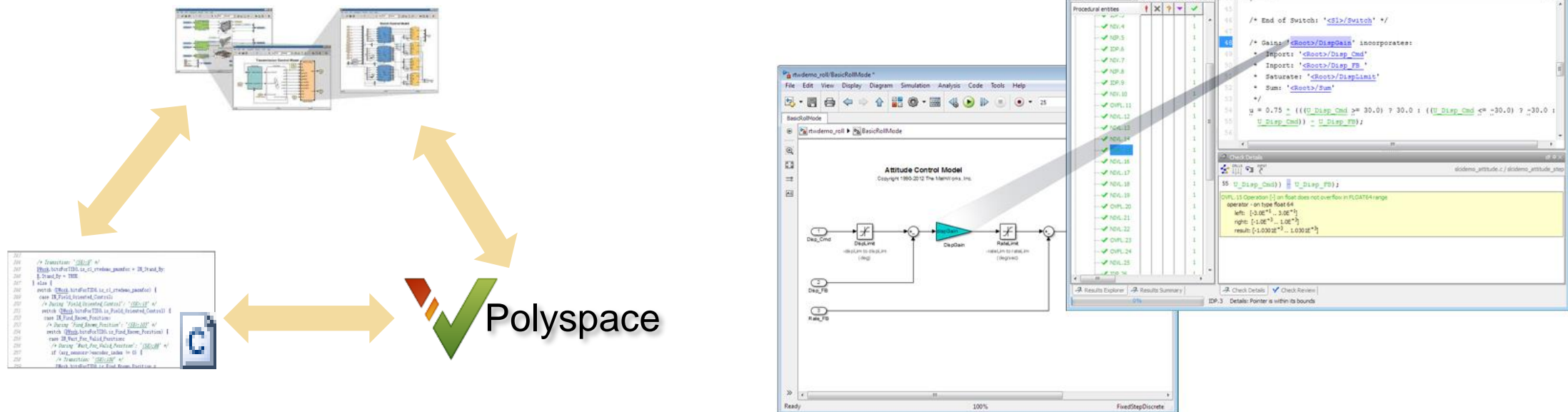
Polyspace Code Prover

MISRAルール違反の正当化の
根拠としてPolyspaceを活用可能

MathWorksツールチェーンへの統合

PolyspaceのSimulink連携機能でモデル修正が容易

- Simulink®環境からPolyspace解析を実行
- Polyspaceで発見した不具合をSimulinkモデルで強調表示
- モデルの入力・パラメータ情報を利用



アジェンダ

第1部 静的コード解析手法を提供するPolyspace紹介

第2部 Polyspaceを利用した静的コード解析



2.1 Polyspaceプロジェクト作成

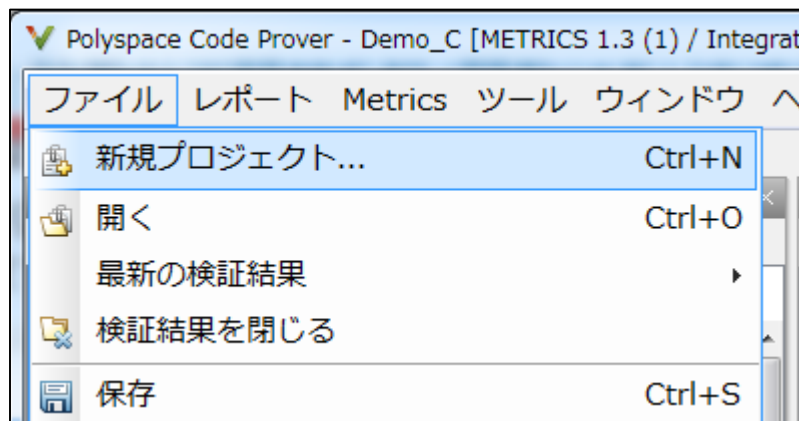
2.2 レビュー時のTips & 便利機能

2.3 まとめ

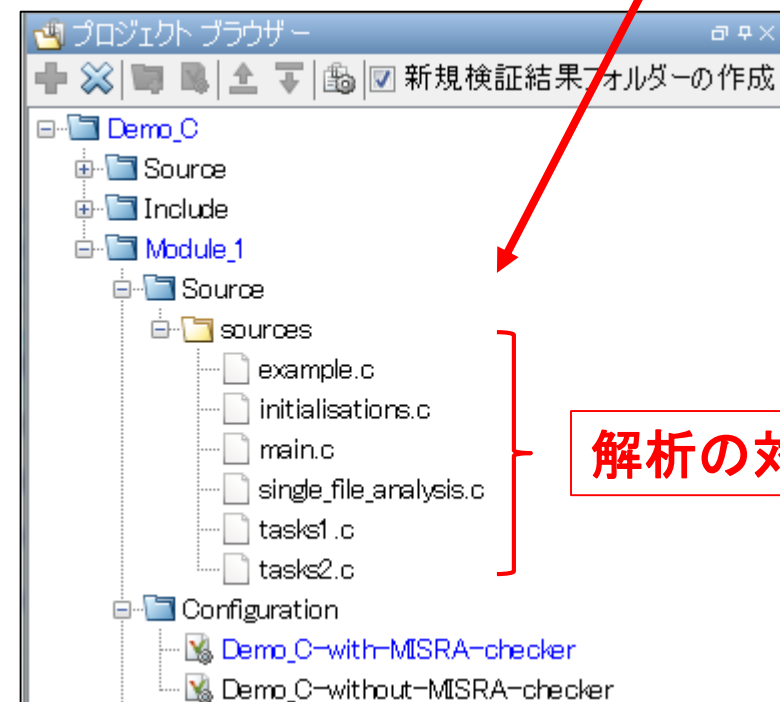
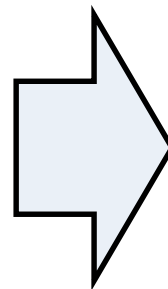
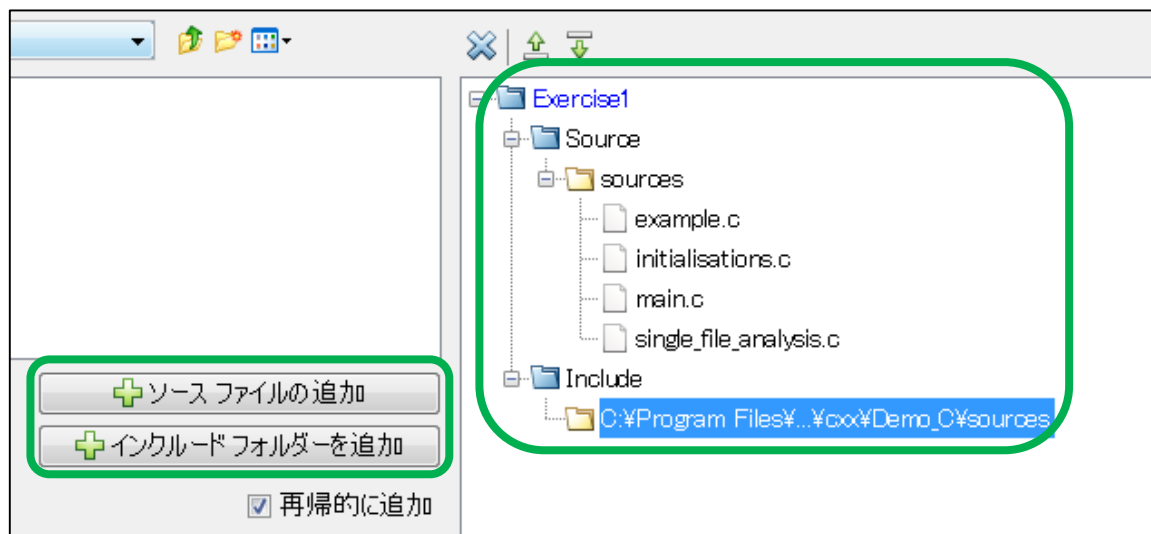
第3部 Q&A会

Polyspace Code Proverプロジェクトの作成

- 新規プロジェクトの作成
- ソースファイルとインクルードディレクトリーの追加

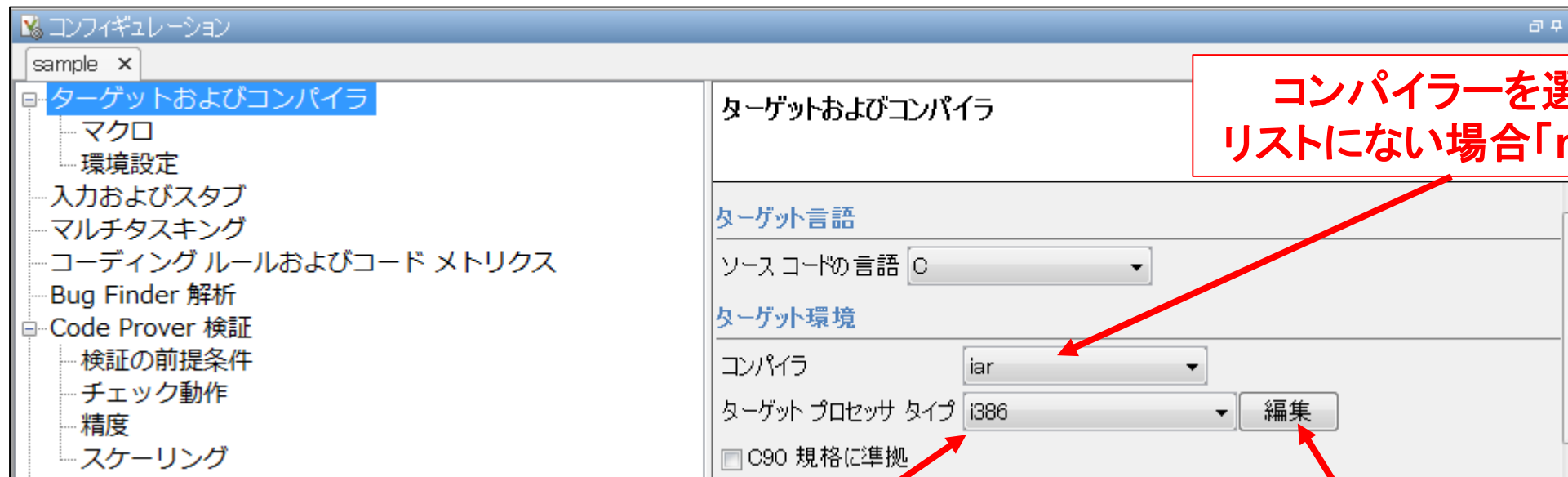


ドラッグ & ドロップ可能



解析の対象

Polyspace設定:ターゲットおよびコンパイラ

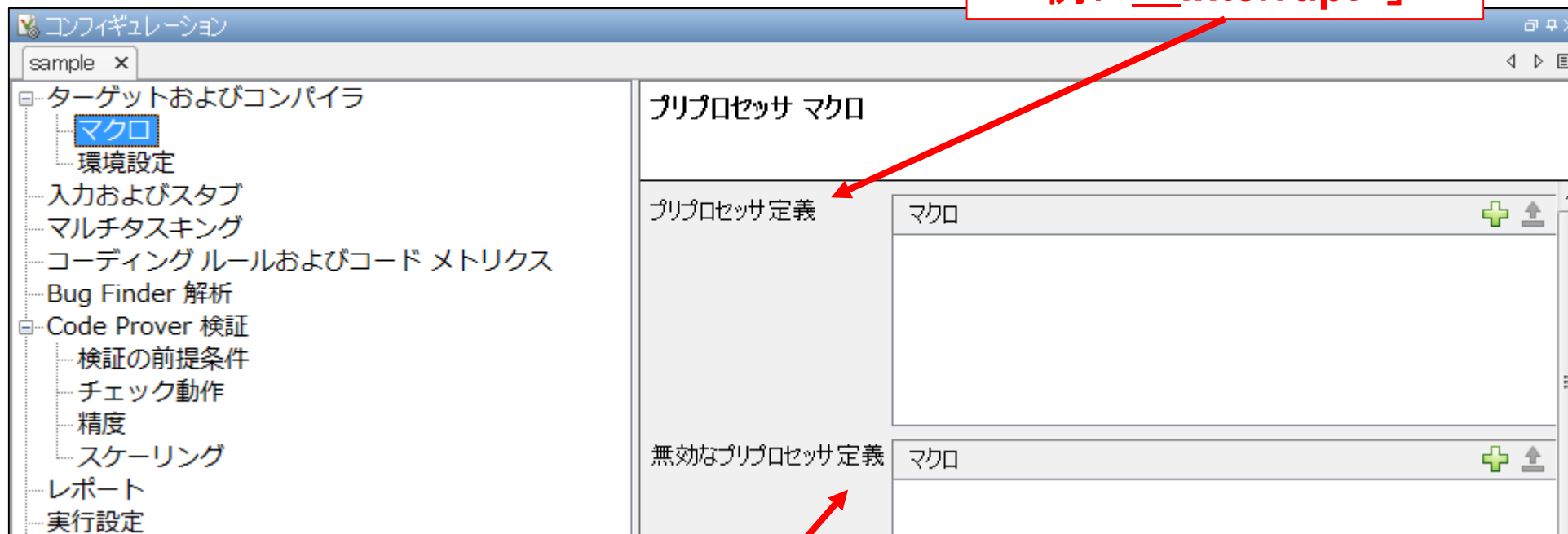


コンパイラを選択
リストにない場合「none」

マイコンタイプを選択

- ・データ型のビット数を確認
- ・カスタマイズ可能

Polyspace設定: マクロ



前処理ディレクティブ定義
例:「__interrupt=」

定義を無効にする
プリプロセッサ定義

Polyspace設定: 入力およびスタブ

グローバル変数、スタブ関数出力値の範囲を指定

The screenshot displays the Polyspace Configuration window with the 'Inputs and Stubs' tab selected. The left sidebar shows the configuration tree, with 'Inputs and Stubs' highlighted. The main area is divided into 'Inputs' and 'Stubs' sections. A red box highlights the text 'グローバル変数、スタブ関数出力値の範囲を指定' (Specify the range of output values for global variables and stub functions), with red arrows pointing to the 'Inputs' section and the 'Edit' button.

Inputs and Stubs Configuration

- ターゲットおよびコンパイラ
 - マクロ
 - 環境設定
 - 入力およびスタブ**
 - マルチタスキング
 - コーディング ルールおよびコード
 - Bug Finder 解析
- Code Prover 検証
 - 検証の前提条件
 - チェック動作
 - 精度
 - スケーリング
- レポート
- 実行設定
- 詳細設定

入力 (Inputs)

制約の設定 **編集**

☐ グローバル変数の既定の初期化を無視する

スタブ (Stubs)

スタブを生成する関数

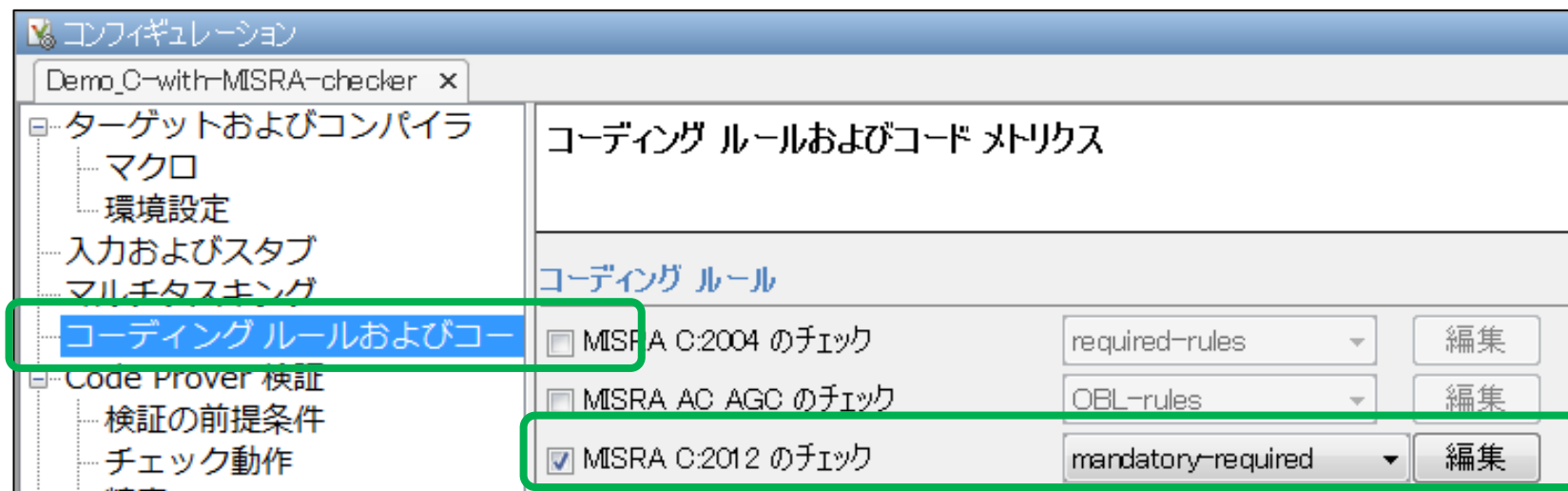
| 関数 |
|----|
|----|

制約指定 - C:\Polyspace_work\Examples\R2017a\Code_Prover_Example\Module_1_Code_Prover_Example_drs_template.xml

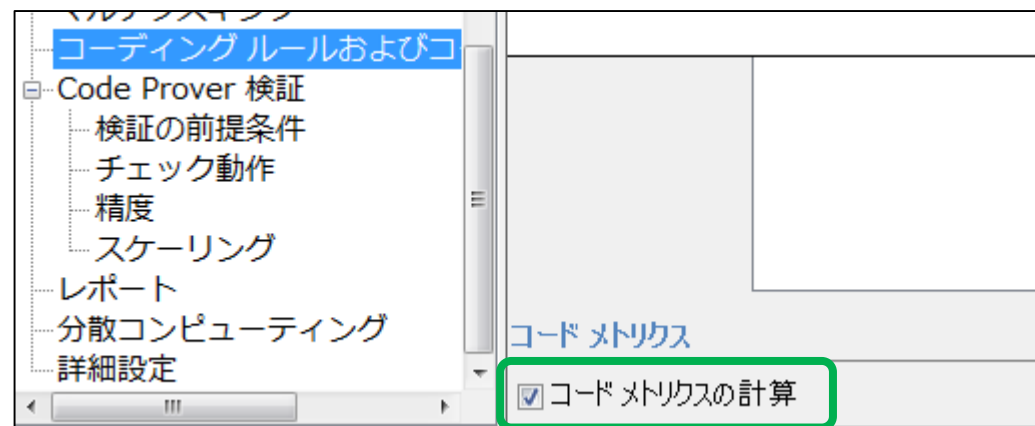
| 名前 | ファイル | 属性 | データ型 | main ジェネレーターによる... | 初期化モード | 初期化範囲 |
|------------------|------------------------|--------|-------------|--------------------|----------------|-------|
| Global Variables | | | | | | |
| PowerLevel | task1.c | | int32 | | MAIN GENERATOR | |
| SHR | task1.c | static | int32 | | MAIN GENERATOR | |
| SHR2 | task1.c | static | int32 | | MAIN GENERATOR | |
| SHR4 | task1.c | static | | | | |
| SHR5 | task1.c | static | int32 | | MAIN GENERATOR | |
| SHR6 | task1.c | static | int32 | | MAIN GENERATOR | |
| _huge_val | huge_val.h | static | | | | |
| _huge_valf | huge_valf.h | static | | | | |
| _huge_vall | huge_vall.h | static | | | | |
| _nan_union | nan.h | static | | | | |
| arr | initialisations.c | | int32 * | | MAIN GENERATOR | |
| current_data | initialisations.c | static | int32 * | | MAIN GENERATOR | |
| first_payload | initialisations.c | | int32 | | MAIN GENERATOR | |
| output_v1 | single_file_analysis.c | static | int8 | | MAIN GENERATOR | |
| output_v6 | single_file_analysis.c | static | int32 | | MAIN GENERATOR | |
| output_v7 | single_file_analysis.c | static | int32 | | MAIN GENERATOR | |
| saved_values | single_file_analysis.c | static | int16 [127] | | | |
| second_payload | initialisations.c | | int32 | | MAIN GENERATOR | |
| tab | initialisations.c | | int32 [10] | | | |

Polyspace設定:MISRA C:2012ルールチェッカー

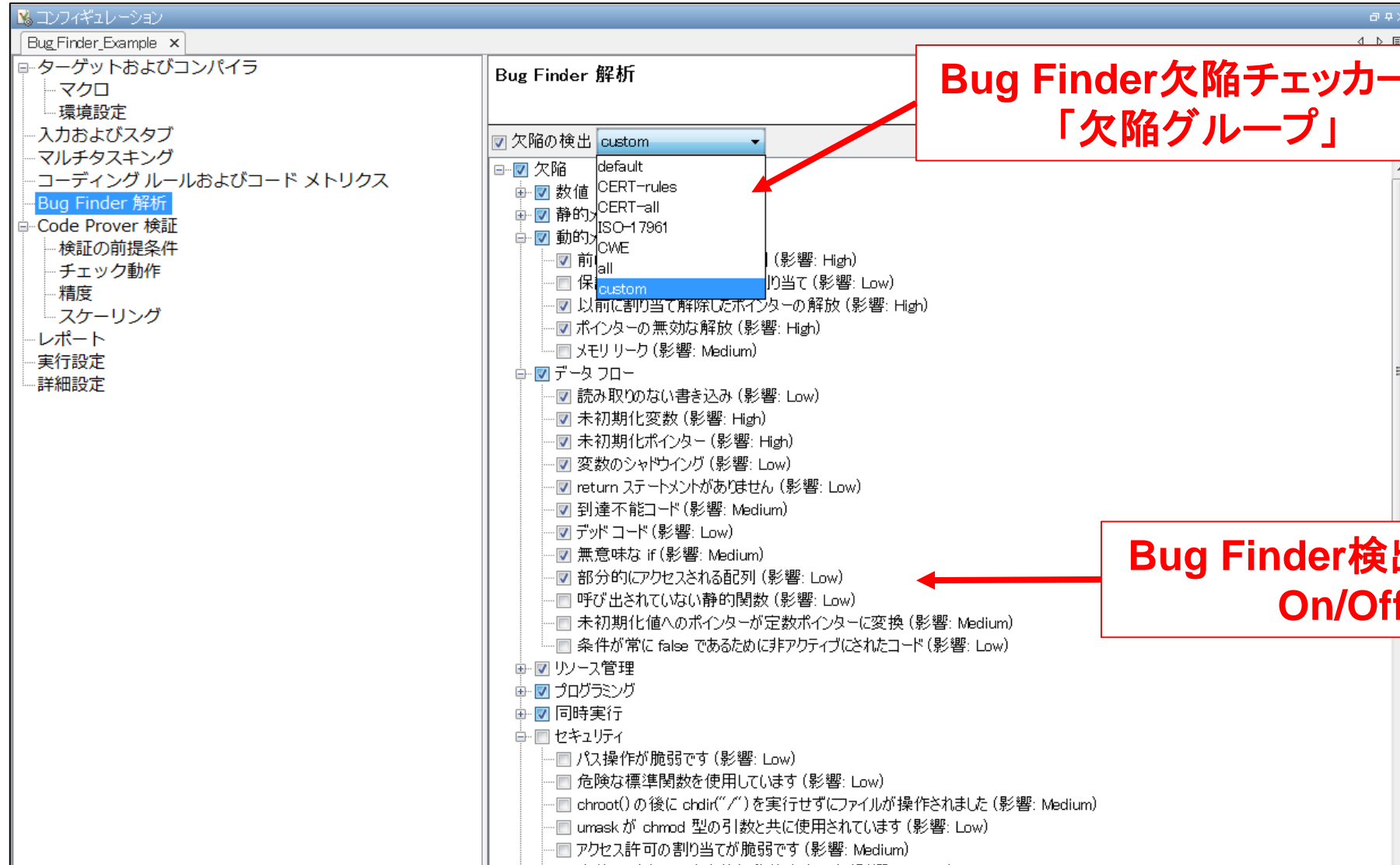
- コーディングルールおよびコードメトリクス→MISRA C:2012



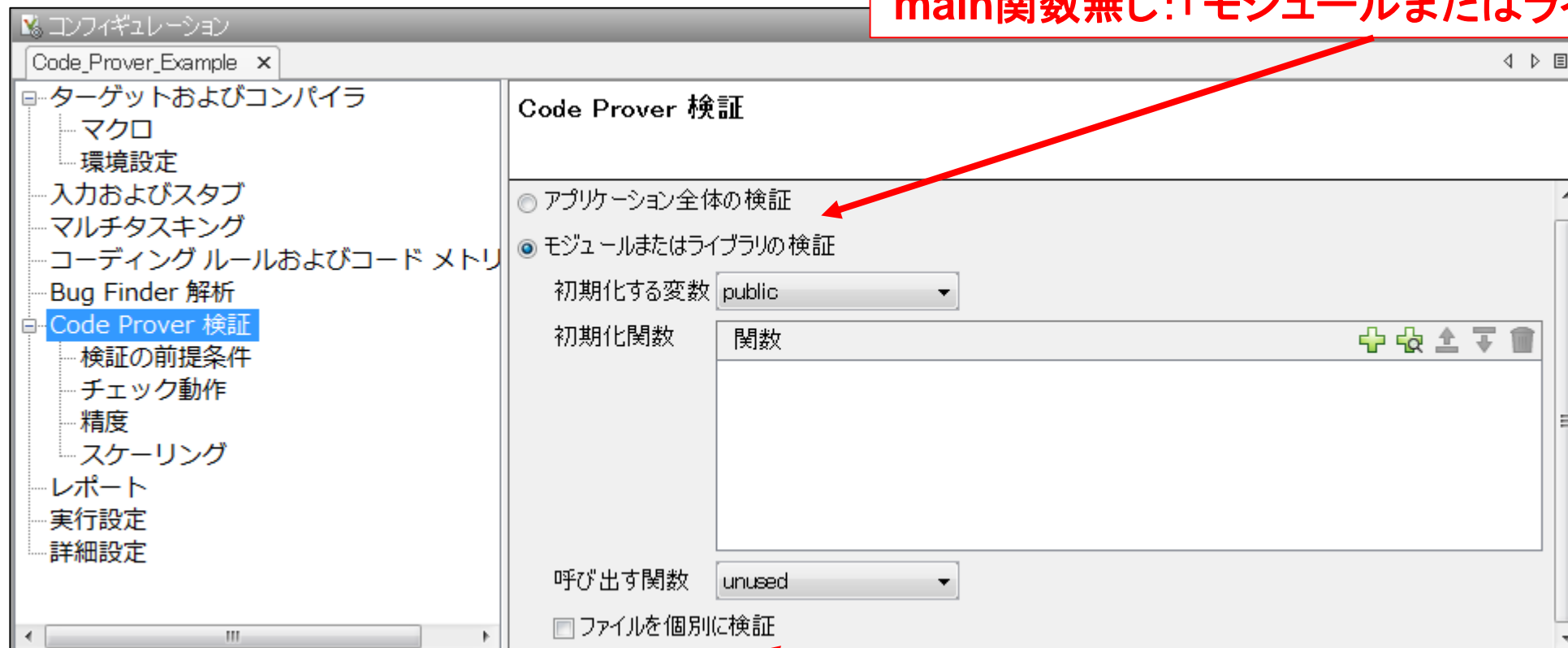
- コードメトリクスの計算



Polyspace設定: Bug Finder解析



Polyspace設定: Code Prover検証

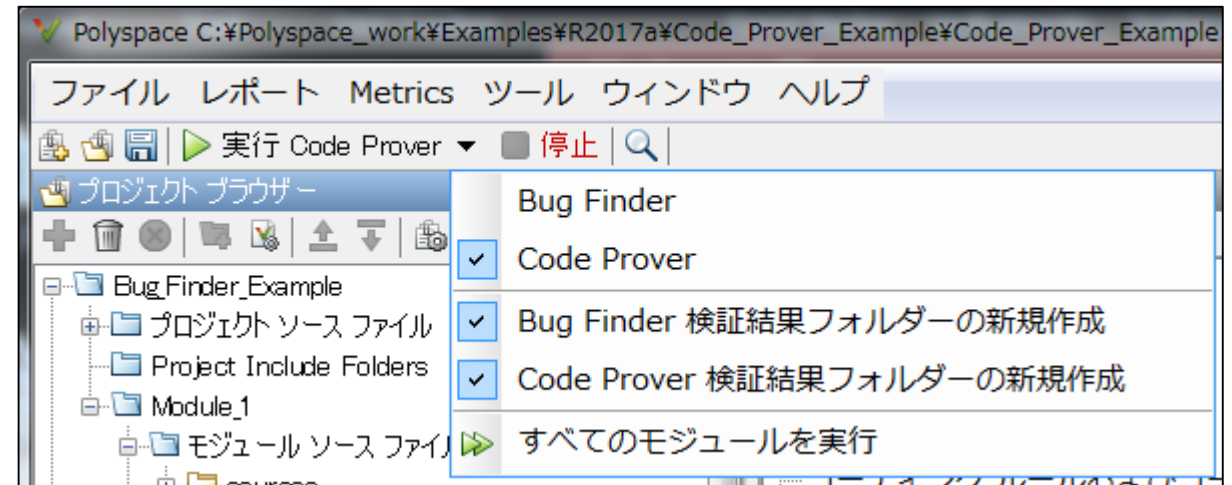


main関数有り:「アプリケーション全体の検証」
main関数無し:「モジュールまたはライブラリの検証」

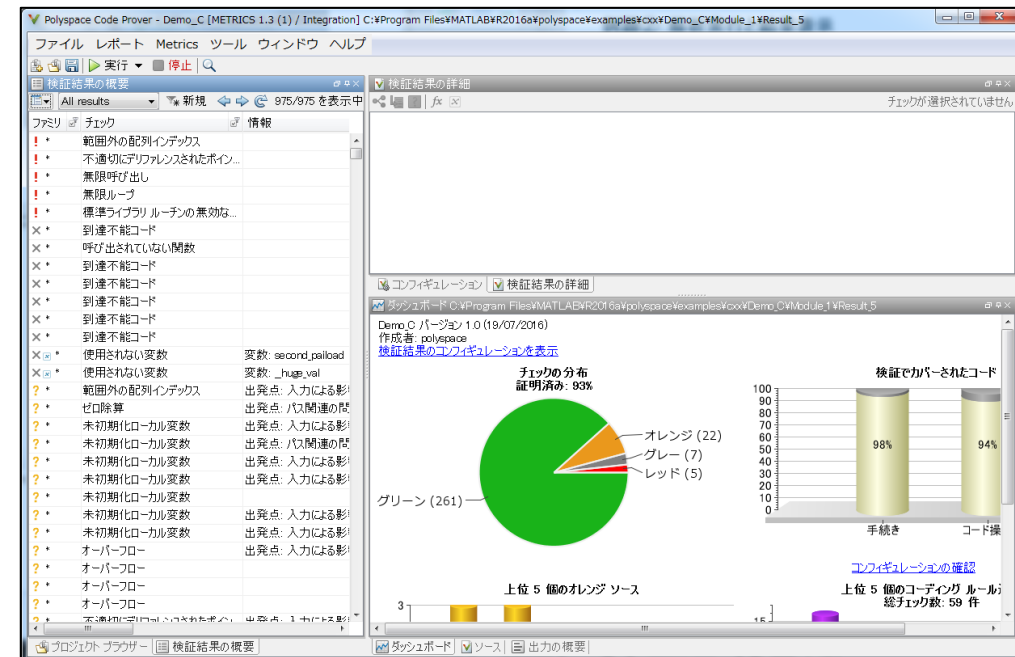
ファイルを別々に解析

解析実行と結果表示

- Polyspace Code ProverかPolyspace Bug Finderを選択し、検証を実行



- 解析終了後に結果が開きます



コンパイルフェーズでエラーが発生...

- 「出力の概要」タブを確認
- コンパイルエラーに関するメッセージを確認
- エラーを右クリックし、ソースコードを確認

出力の概要 C:\Polyspace_work\sample\Module_5\CP_Result_1

✖ コンパイル エラー: 12 | フィルター-警告 (0)

| 型 | メッセージ | ファイル | 行 |
|---|--|-------------|----|
| ✖ | identifier "u2" is undefined | MM_sample.c | 3 |
| ✖ | expected a ";" | MM_sample.c | 3 |
| ✖ | "u2" is not a type name | MM_sample.c | 4 |
| ✖ | expected a ";" | MM_sample.c | 4 |
| ✖ | "u2" is not a type name | MM_sample.c | 5 |
| ✖ | expected a ";" | MM_sample.c | 5 |
| ✖ | "u2" is not a type name | MM_sample.c | 6 |
| ✖ | expected a ";" | MM_sample.c | 6 |
| ✖ | "u2" is not a type name | MM_sample.c | 7 |
| ✖ | expected a ";" | MM_sample.c | 7 |
| ✖ | expected a ";" | MM_sample.c | 9 |
| ✖ | identifier "u2t_ReadBuff" is undefined | MM_sample.c | 11 |

エディターを開く
前処理済みファイルを開く
エディターを指定


ダッシュボード | ソース | 出力の概要 | 実行ログ

アジェンダ

第1部 静的コード解析手法を提供するPolyspace紹介

第2部 Polyspaceを利用した静的コード解析

2.1 Polyspaceプロジェクト作成

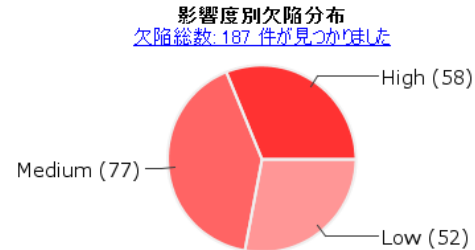
 2.2 レビュー時のTips & 便利機能

2.3 まとめ

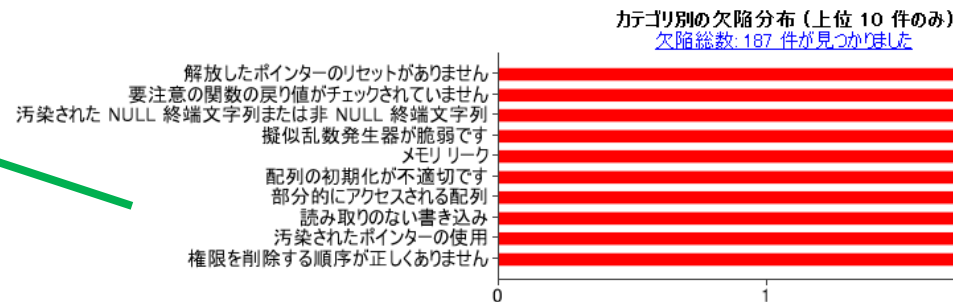
第3部 Q&A会

Polyspace Bug Finder 解析結果のレビュー

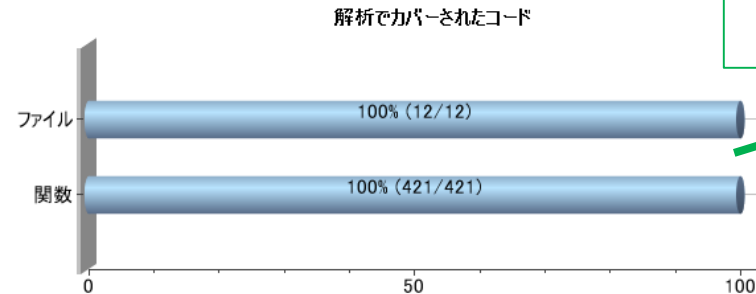
欠陥の影響度



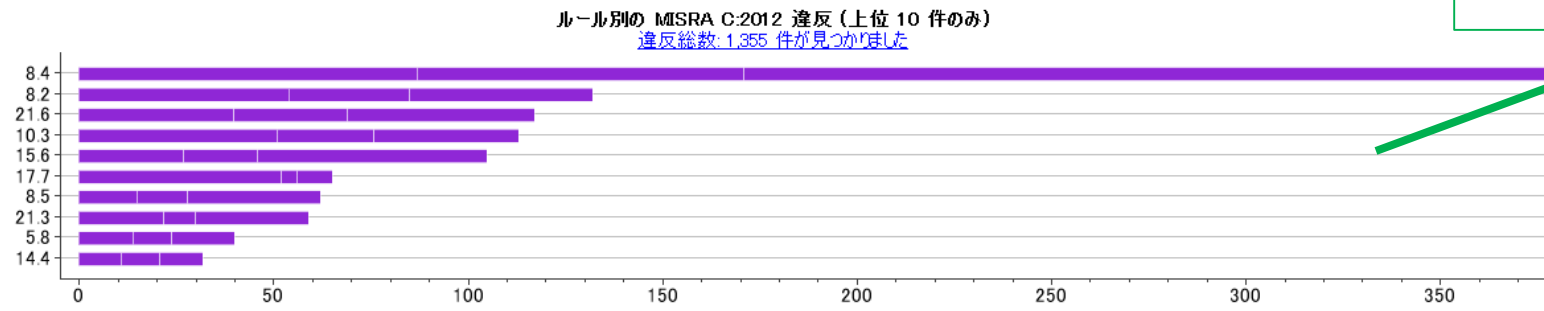
上位10件の欠陥種類



解析のカバー率



上位10件のMISRAルール違反



Polyspace Bug Finder 解析結果のレビュー

Polyspace Bug Finder - Exercise1 C:\Polyspace_work\Exercise1\Result_Exercise1

ファイル レポート Metrics ツール ウィンドウ ヘルプ

実行 停止

検証結果の概要

All results 新規 409/409 を表示中

ファミリー 情報 ファイル

欠陥 24

- データフロー 15
 - デッドコード 2
 - 呼び出されていない静的関数 1
- 未初期化変数 2
 - 影響: High initialisations.c
 - 影響: High initialisations.c
- 読み取りのない書き込み 9
- 部分的にアクセスされる配列 1
- 数値 2
- 汚染されたデータ 1
- 適切な手法 3
- 静的メモリ 3
- MISRA C:2012 30
 - 2 Unused code 10
 - 8 Declarations and definitions 2
 - 9 Initialization 1
 - 10 The essential type model 10
 - 13 Side effects 1
 - 14 Control statement expressions 2
 - 17 Functions 4
- コードメトリクス 355
 - プロジェクトメトリクス 1
 - ファイルメトリクス 16
 - 関数メトリクス 338

検証結果の詳細

変数トレース initialisations.c / polynomial()

検証結果のレビュー

重大度 ステータス

ここにコメントを入力...

未初期化変数 (影響: High) ?
Local variable 'y' may be read before being initialized.
The function's known input values will not cause a defect.

| イベント | ファイル | スコープ | 行 |
|-------------------------------|-------------------|--------------|----|
| 1 Declaration of variable 'y' | initialisations.c | polynomial() | 76 |
| 2 Not entering for loop | initialisations.c | polynomial() | 78 |
| 3 未初期化変数 | initialisations.c | polynomial() | 82 |

コンフィギュレーション 検証結果の詳細 コンテキストヘルプ

ソース

```
initialisations.c x
/4 int polynomial(int input)
75 {
76     int y, i;
77
78     for (i = 0; i <= input; i++) {
79         y = degree_computation(1, -23, -15, i);
80     }
81
82     if ((y >= -5) && (y <= 7)) {
83         return y;
84     } else {
85         y = return_code(10);
86         return y;
87     }
88 }
```

欠陥リスト

MISRA
ルール違反

コードメトリクス

レビューコメント

詳細

ソースコード

Polyspace Code Prover 検証結果のレビュー



Polyspace Code Prover 検証結果のレビュー

Polyspace Code Prover - Example_Project C:\Polyspace_work\Example_Project\Module_1\Result_1

ファイル レポート Metrics ツール ウィンドウ ヘルプ

実行 停止

検証結果の概要

All results 新規 651/651 を表示中

ファミリー チェック 情報

不適切にデリファレンスされたポイン...

無限呼び出し

無限ループ

標準ライブラリ ルーチンの無効な...

到達不能コード

到達不能コード

到達不能コード

到達不能コード

到達不能コード

到達不能コード

使用されない変数 変数: second_payload

使用されない変数 変数: _huge_val

範囲外の配列インデックス 出発点: 入力による影

ゼロ除算 出発点: パス関連の問

未初期化ローカル変数 出発点: 入力による影

未初期化ローカル変数 出発点: 入力による影

未初期化ローカル変数 出発点: 入力による影

未初期化ローカル変数 出発点: 入力による影

未初期化ローカル変数 出発点: 入力による影

未初期化ローカル変数 出発点: 入力による影

オーバーフロー 出発点: 入力による影

オーバーフロー 出発点: 入力による影

不適切にデリファレンスされたポイン... 出発点: 入力による影

不適切にデリファレンスされたポイン... 出発点: 入力による影

オーバーフロー 出発点: 入力による影

オーバーフロー 出発点: 入力による影

オーバーフロー 出発点: 入力による影

ユーザー アサーション

ユーザー アサーション

検証結果の詳細

ここにコメントを入力...

重重大度

ステータス

不適切にデリファレンスされたポインタ

Error: pointer is outside its bounds

Dereference of local pointer 'p' (pointer to int 32, size: 32 bits):

Pointer is not null.

Points to 4 bytes at offset 400 in buffer of 400 bytes, so is outside bounds.

Pointer may point to variable or field of variable:

'Pointer_Arithmetic.array'.

コンフィギュレーション 検証結果の詳細

ソース

```
example.c x main.c x single_file_analysis.c x
94 for (i = 0; i < 100; i++) {
95     *p = 0;
96     p++;
97 }
98
99 if (get_bus_status() > 0) {
100     if (get_oil_pressure() > 0) {
101         *p = 5; /* Out of bounds */
102     } else {
103         i++;
104     }
105 }
106
107 i = get_bus_status();
108
```

レビュー
コメント

項目リスト

チェック項目
の詳細

ソースコード

検証結果リストの表示

- 一覧の表示
- 項目タイプでのグルーピング
- ファイルでのグルーピング

| ファミリー | 情報 | ファイル |
|----------------------------------|--------|------|
| Run-time Check | 4 5 19 | |
| + レッド チェック | 4 | |
| + グレー チェック | 5 | |
| + オレンジ チェック | 19 | |
| + グリーン チェック | | |
| グローバル変数 | 2 | |
| + 非共有 | 2 | |
| MISRA C:2012 | | |
| + 4 Code design | | |
| + 8 Declarations and definitions | | |

| | |
|------------------------|----------------|
| -example.c | 3 2 8 83 7 134 |
| + Close_To_Zero() | 3 10 13 |
| + get_oil_pressure() | 2 2 2 13 |
| + Non_Infinite_Loop() | 11 13 |
| + Pointer_Arithmetic() | 1 1 1 19 2 13 |
| + Recursion_caller() | 1 4 13 |
| + Recursion() | 1 14 2 13 |
| + RTE() | 3 13 |
| + Square_Root_conv() | 8 13 |
| + Square_Root() | 1 4 1 13 |
| + Unreachable_Code() | 1 1 8 13 |
| + ファイル スコープ | 4 |
| huge_val.h | 1 |
| + _init_globals() | 1 |

| 検証結果の概要 | | |
|-------------------------------------|---------------|-----------------------------------|
| ▼ | ▼ All results | * 新規 |
| 651/651 を表示中 | | |
| 検証結果を整理するオプションを選択します | | |
| <input checked="" type="checkbox"/> | なし | |
| <input type="checkbox"/> | ファミリー | |
| <input type="checkbox"/> | ファイル | |
| × | * | 到達不能コード |
| × | * | 到達不能コード |
| × | * | 到達不能コード |
| × | * | 到達不能コード |
| × | * | 到達不能コード |
| × | * | 使用されない変数 変数: second_payload |
| × | * | 使用されない変数 変数: _huge_val |
| ? | * | 範囲外の配列インデックス 出発点: 入力による影響 |
| ? | * | ゼロ除算 出発点: パス関連の問題 |
| ? | * | 未初期化ローカル変数 出発点: 入力による影響 |
| ? | * | 未初期化ローカル変数 出発点: 入力による影響 |
| ? | * | 未初期化ローカル変数 出発点: 入力による影響 |
| ? | * | 未初期化ローカル変数 出発点: 入力による影響 |
| ? | * | 未初期化ローカル変数 出発点: 入力による影響 |
| ? | * | 未初期化ローカル変数 出発点: 入力による影響 |
| ? | * | オーバーフロー 出発点: 入力による影響 |
| ? | * | オーバーフロー 出発点: 入力による影響 |
| ? | * | 不適切にデリファレンスされたポイン... 出発点: 入力による影響 |
| ? | * | 不適切にデリファレンスされたポイン... 出発点: 入力による影響 |
| ? | * | オーバーフロー 出発点: 入力による影響 |

グリーンチェック項目

グリーン = 安全

functional_ranges() と
new_speed()は安全

```

56 return (in / 9 + ((s32)ex_speed + (s32)c_speed) / 2);
57 }
58
59
60 static char rese

```

operator / on type int 32
left: [-1701 .. 3276]
right: 9
result: [-189 .. 364]

| 検証結果の概要 | | | | |
|-------------------------|--------------|----|---|-------|
| Checks & Rules | | | | |
| ファミリ | チェック | 情報 | | |
| -main.c | | 1 | 1 | 3 9 7 |
| + interpolation() | | 1 | 1 | 1 7 1 |
| + main() | | 2 | 2 | 6 |
| -single_file_analysis.c | | 2 | 8 | 85 12 |
| + all_values_s16() | | 2 | 5 | 2 |
| + all_values_s32() | | 2 | 5 | 2 |
| + all_values_u16() | | 2 | 5 | 2 |
| + functional_ranges() | | 6 | | |
| ✓ * | 初期化されていない戻り値 | | | |
| ✓ * | 初期化されていない戻り値 | | | |
| ✓ * | 初期化されていない戻り値 | | | |
| ✓ * | 初期化されていない戻り値 | | | |
| ✓ * | 初期化されていない戻り値 | | | |
| ✓ * | 初期化されていない戻り値 | | | |
| + generic_validation() | | 2 | 2 | 51 2 |
| + new_speed() | | 9 | | |
| ✓ * | 未初期化ローカル変数 | | | |
| ✓ * | オーバーフロー | | | |
| ✓ * | ゼロ除算 | | | |
| ✓ * | オーバーフロー | | | |
| ✓ * | 未初期化ローカル変数 | | | |
| ✓ * | オーバーフロー | | | |
| ✓ * | 未初期化ローカル変数 | | | |
| ✓ * | オーバーフロー | | | |
| ✓ * | ゼロ除算 | | | |
| + reset_temperature() | | 4 | 2 | |

グレーチェック項目

グレー = 非到達コード

generic_validation()に
非到達コード有り

検証結果の概要

Checks & Rules *新規

| ファミリ | チェック | 情報 |
|-------------------------|--------------------------------|----------------|
| -main.c | | 1 1 3 9 7 |
| +interpolation() | | 1 1 1 7 1 |
| +main() | | 2 2 6 |
| -single_file_analysis.c | | 2 8 85 12 |
| +all_values_s16() | | 2 5 2 |
| +all_values_s32() | | 2 5 2 |
| +all_values_u16() | | 2 5 2 |
| +functional_ranges() | | 6 |
| ✓ * | 初期化されていない戻り値 | |
| ✓ * | 初期化されていない戻り値 | |
| ✓ * | 初期化されていない戻り値 | |
| ✓ * | 初期化されていない戻り値 | |
| ✓ * | 初期化されていない戻り値 | |
| ✓ * | 初期化されていない戻り値 | |
| -generic_validation() | | 2 2 51 2 |
| ✗ * | 到達不能コード | |
| ✗ * | 到達不能コード | |
| ? * | 範囲外の配列インデックス | 出発点: 入力によ |
| ? * | オーバーフロー | 出発点: 入力によ |
| ▽ * | 18.1 A pointer resulting fr... | カテゴリ: Required |
| ▽ * | 10.3 The value of an expr... | カテゴリ: Required |
| ✓ * | オーバーフロー | |
| ✓ * | 未初期化変数 | |

```

87 v0_c = (s16) gVALUE(v0);
88 if (v0_c == 90) {
89     retu
90     /* g
91     * w
92     * i

```

Local variable 'v0_c' (int 16): [0 .. 26]
Conversion from int 16 to int 32
right: [0 .. 26] ted to [0 ; 26624]
result: [0 .. 26] 0 = 90 * BIN_v0)

レッド = エラー

Pointer_Arithmetic()にエラー有り

検証結果の概要

Checks & Rules 新規

| ファミリ | チェック | 情報 |
|------------------------|--------------------------------|----------------|
| example.c | | 3 2 8 83 7 |
| + Close_To_Zero() | | 3 10 |
| + get_oil_pressure() | | 2 2 2 |
| + Non_Infinite_Loop() | | 11 |
| + Pointer_Arithmetic() | | 1 1 10 2 |
| ! * | 不適切にデリファレンスされた... | |
| X * | 到達不能コード | |
| ? * | 不適切にデリファレンスされた... | 出発点: 入力によ |
| ▽ * | 18.1 A pointer resulting fr... | カテゴリ: Required |
| ▽ * | 18.1 A pointer resulting fr... | カテゴリ: Required |
| ✓ * | 未初期化ローカル変数 | |
| ✓ * | 未初期化ローカル変数 | |
| ✓ * | オーバーフロー | |
| ✓ * | 不適切にデリファレンスされた... | |
| ✓ * | インター | |
| ✓ * | インター | |
| ✓ * | いない戻り値 | |
| ✓ * | いない戻り値 | |
| ✓ * | インター | |

r to int 32, size: 32 bits):

uffer of 400 bytes, so is outside bounds.

```

99  if (get_bus_status() > 0) {
100      if (get_oil_pressure() > 0) {
101          *p = 5; /* Out of bounds */
102      } else
103          i
104      }
105  }
106

```

Dereference of local pointer 'p' (pointer to int 32, size: 32 bits):

- Pointer is not null.
- Points to 4 bytes at offset 400 in buffer of 400 bytes, so is outside bounds.
- Pointer may point to variable or field of variable: 'Pointer_Arithmetic:array'.

オレンジチェック項目

オレンジ = エラーが発生するパターンが検出

generic_validation()に
範囲外の配列インデックス
が検出

検証結果の概要

Checks & Rules *新規

| ファミリー | チェック | 情報 |
|------------------------|----------------------------------|----------------|
| main.c | interpolation() | 1 1 3 9 7 |
| | main() | 1 1 1 7 1 |
| single_file_analysis.c | | 2 2 6 |
| | all_values_s16() | 2 8 85 12 |
| | all_values_s32() | 2 5 2 |
| | all_values_u16() | 2 5 2 |
| | functional_ranges() | 6 |
| | generic_validation() | 2 2 51 2 |
| | * 到達不能コード | |
| | * 到達不能コード | |
| | * 範囲外の配列インデックス | 出発点: 入力による |
| | * オーバーフロー | 出発点: 入力による |
| | * 18.1 A pointer resulting fr... | カテゴリ: Required |
| | * 10.3 The value of an expr... | カテゴリ: Required |

```

123 if (output_v7 >= 0) {
124     saved_values[output_v7] = s8_ret;
125     return s8_ret;
126 }
127 return reset_temp;
128 }

```

Assignment to element of static array (int 16): [-32 .. 112]
array size: 127
array index value: [0 .. 555]

Press 'F2' for focus

変数アクセス: グローバル・共有変数のレビュー

テーブルビュー

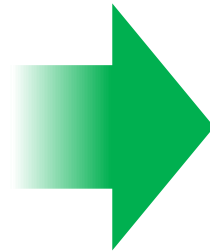
Orange: unproven
may be unsafe shared
memory access

Green: reliable
safe shared memory access

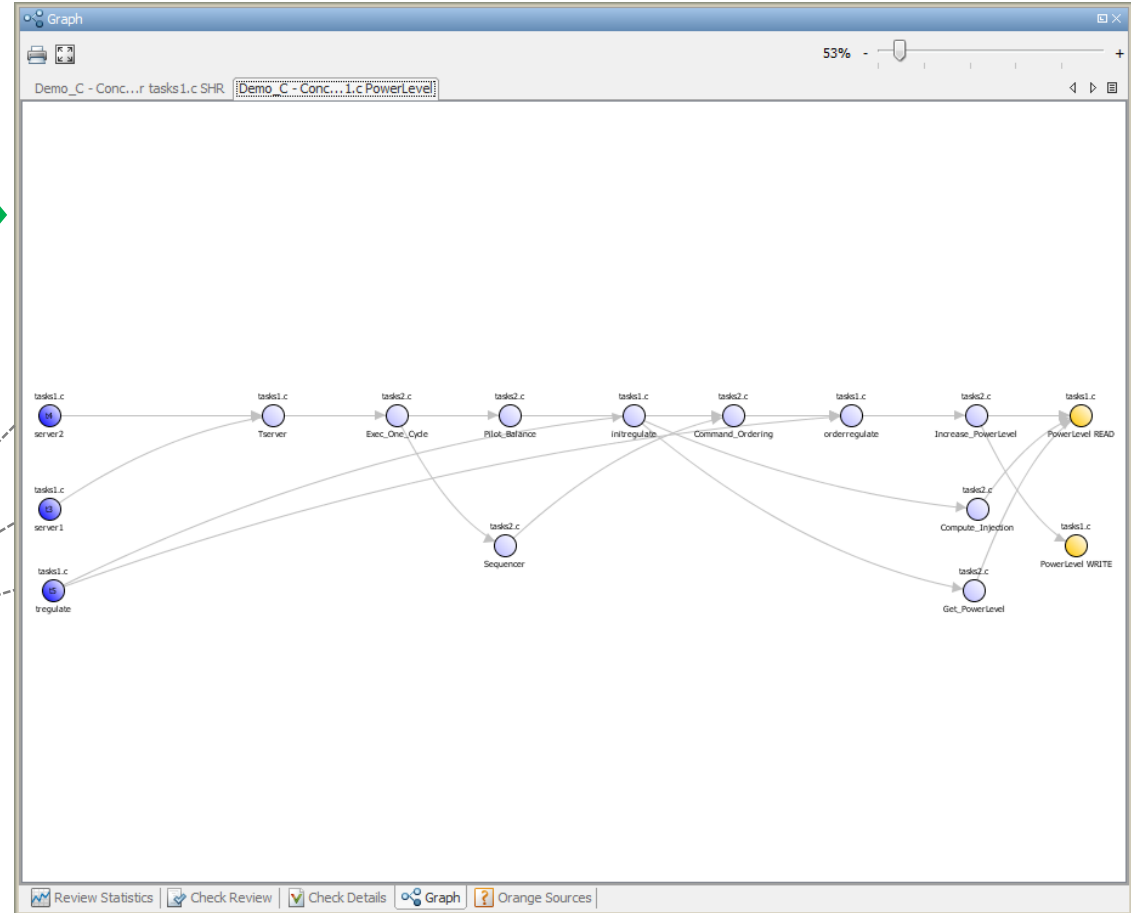
| Variables | Values | # Re... | # Wri... | Wri... | Re... | Protection | Usage | Scalar | Line | Col | File | Type | Detailed T... |
|-----------------------------------|--------------------------|---------|----------|----------|----------|------------------|--------|--------|------|-----|-------------------|-----------------|---------------|
| initialisations.current_data | | 2 | 2 | | | | | | 8 | 12 | initialisation... | pointer to i... | |
| initialisations.first_paload | 100 | 0 | 3 | | | | | | 13 | 4 | initialisation... | int 32 | |
| initialisations.second_paload | 200 | 0 | 1 | | | | | | 14 | 4 | initialisation... | int 32 | |
| initialisations.tab | 0 or 12 | 1 | 3 | | | | | | 10 | 4 | initialisation... | array(0..9)... | |
| single_file_analysis.output_v1 | [-31 .. 127] | 0 | 2 | | | | | | 24 | 10 | single_file_... | int 8 | |
| single_file_analysis.output_v6 | [-1701] | 1 | 3 | | | | | | 22 | 11 | single_file_... | int 32 | |
| single_file_analysis.output_v7 | [-253] | 3 | 2 | | | | | | 23 | 11 | single_file_... | int 32 | |
| single_file_analysis.saved_values | [-32 .. 112] | 0 | 2 | | | | | | 26 | 11 | single_file_... | array(0..1... | |
| single_file_analysis.v0 | [0 .. 266... | 1 | 2 | | | | | | 14 | 11 | single_file_... | unsigned in... | |
| single_file_analysis.v1 | [0 .. 230... | 3 | 2 | | | | | | 15 | 11 | single_file_... | int 16 | |
| single_file_analysis.v2 | [-25920] | 1 | 2 | | | | | | 16 | 11 | single_file_... | int 16 | |
| single_file_analysis.v3 | [0 .. 216] | 2 | 2 | | | | | | 17 | 10 | single_file_... | unsigned int 8 | |
| single_file_analysis.v4 | [-360] | 1 | 2 | | | | | | 18 | 11 | single_file_... | int 16 | |
| single_file_analysis.v5 | [-1440] | 1 | 2 | | | | | | 19 | 11 | single_file_... | int 16 | |
| tasks1.PowerLevel | [-21474836 .. -10000] | 4 | 3 | t3 t4 t5 | t3 t4 t5 | | shared | | 26 | 4 | tasks1.c | int 32 | |
| main.main | | | | | | | | | 36 | 4 | main.c | | |
| tasks1._init_globals | 0 | | | | | | | | 26 | 4 | tasks1.c | | |
| tasks2.Increase_PowerLevel | [-21474836 .. -21474836] | | | | | | | | 19 | 4 | tasks2.c | | |
| tasks1.orderregulate | [-21474836 .. -21474836] | | | | | | | | 40 | 10 | tasks1.c | | |
| tasks2.Increase_PowerLevel | [-21474836 .. -21474836] | | | | | | | | 19 | 4 | tasks2.c | | |
| tasks2.Compute_Injection | [-21474836 .. -21474836] | | | | | | | | 34 | 27 | tasks2.c | | |
| tasks2.Get_PowerLevel | [-21474836 .. -21474836] | | | | | | | | 41 | 10 | tasks2.c | | |
| tasks1.server1 | | | | t3 | | | | | | | | | |
| tasks1.server2 | | | | t4 | | | | | | | | | |
| tasks1.tregulate | | | | t5 | | | | | | | | | |
| tasks1.server1 | | | | | t3 | | | | | | | | |
| tasks1.server2 | | | | | t4 | | | | | | | | |
| tasks1.tregulate | | | | | t5 | | | | | | | | |
| tasks1.SHR | 0 or 22 | 1 | 2 | t3 t4 | t5 | Critical sect... | shared | | 30 | 11 | tasks1.c | int 32 | |
| tasks1.SHR2 | 0 or 22 | 1 | 3 | t3 t4 | t5 | | shared | | 31 | 11 | tasks1.c | int 32 | |
| tasks1.SHR3 | [0 .. 29] ... | 1 | 2 | | | | | | 109 | 15 | tasks1.c | int 32 | |
| tasks1.SHR4 | | 2 | 3 | t2 t3... | t2 t3... | | shared | | 28 | 11 | tasks1.c | struct {A: i... | |
| tasks1.SHR5 | 5 or 28 | 2 | 2 | t1 | t1 t2 | Temporal e... | shared | | 29 | 11 | tasks1.c | int 32 | |
| tasks1.SHR6 | 0 | 2 | 1 | | | | | | 32 | 11 | tasks1.c | int 32 | |

変数アクセス: グローバル・共有変数のレビュー グラフィカルビュー

| Variable Access | |
|----------------------------------|---------|
| Variables | Values |
| initialisations.all | |
| + initialisations.current_data | |
| + initialisations.first_paiload | 100 |
| + initialisations.second_paiload | 200 |
| + initialisations.tab | 0 or 12 |



**Function Call Tree
to shared memory
access**



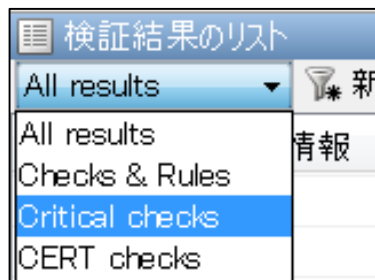
対話的な関数コールツリー

**Complete Call
Tree Hierarchy**

**Functions used
by multiple
tasks/threads**

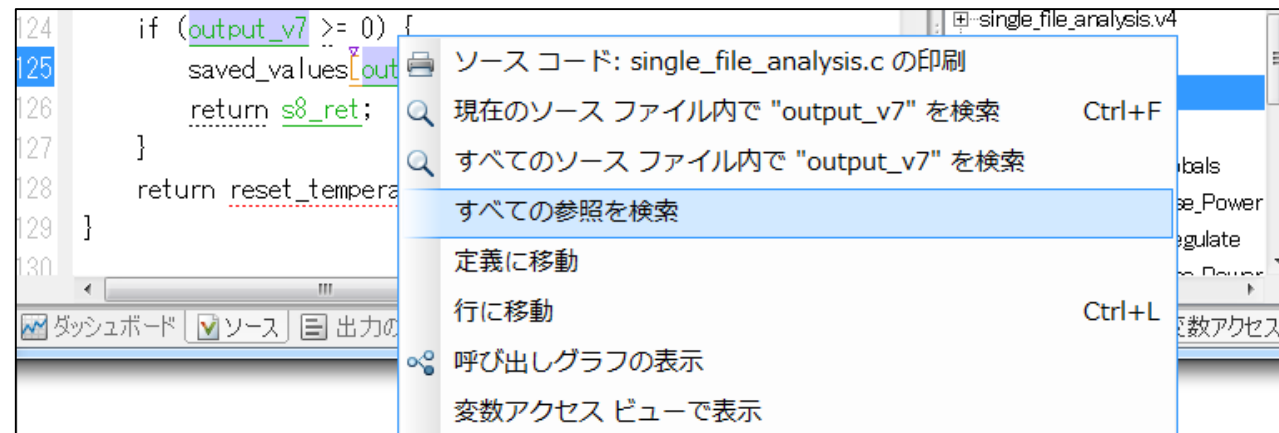
| Calls | Line |
|----------------------------|------|
| tasks1.orderregulate | 35 |
| tasks2.Increase_PowerLevel | 38 |
| tasks2.Command_Ordering | 50 |
| tasks2.Pilot_Balance | 56 |
| tasks2.Exec_One_Cycle | 68 |
| tasks1.Tserver | 83 |
| tasks1.server2 | 91 |
| main.main | 33 |
| tasks1.server1 | 97 |
| tasks1.server2 | 0 |
| tasks1.server1 | 0 |
| tasks1.server2 | 0 |
| tasks1.server1 | 0 |
| tasks1.server2 | 0 |
| tasks1.server1 | 0 |
| tasks2.Sequencer | 62 |
| tasks2.Exec_One_Cycle | 69 |
| tasks1.server2 | 0 |
| tasks1.server1 | 0 |
| tasks1.server2 | 0 |
| tasks1.server1 | 0 |
| tasks1.initregulate | 51 |
| tasks1.tregulate | 68 |
| tasks1.Tserver | 77 |
| tasks1.tregulate | 0 |
| tasks1.server2 | 0 |
| tasks1.server1 | 0 |

その他の便利機能

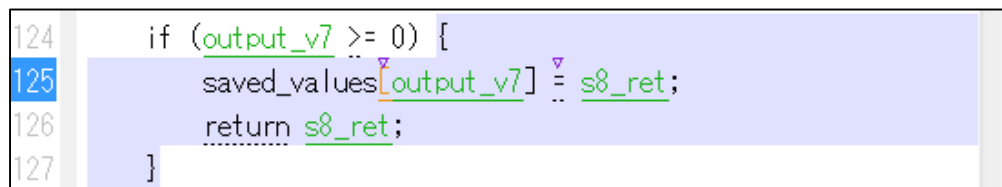


フィルタリング:

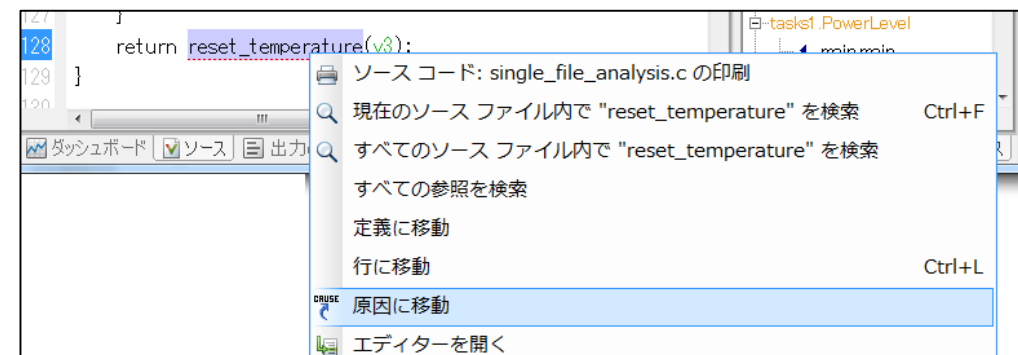
- ・ 新規項目
- ・ クリティカルチェック



検索機能



コードブロックのハイライト

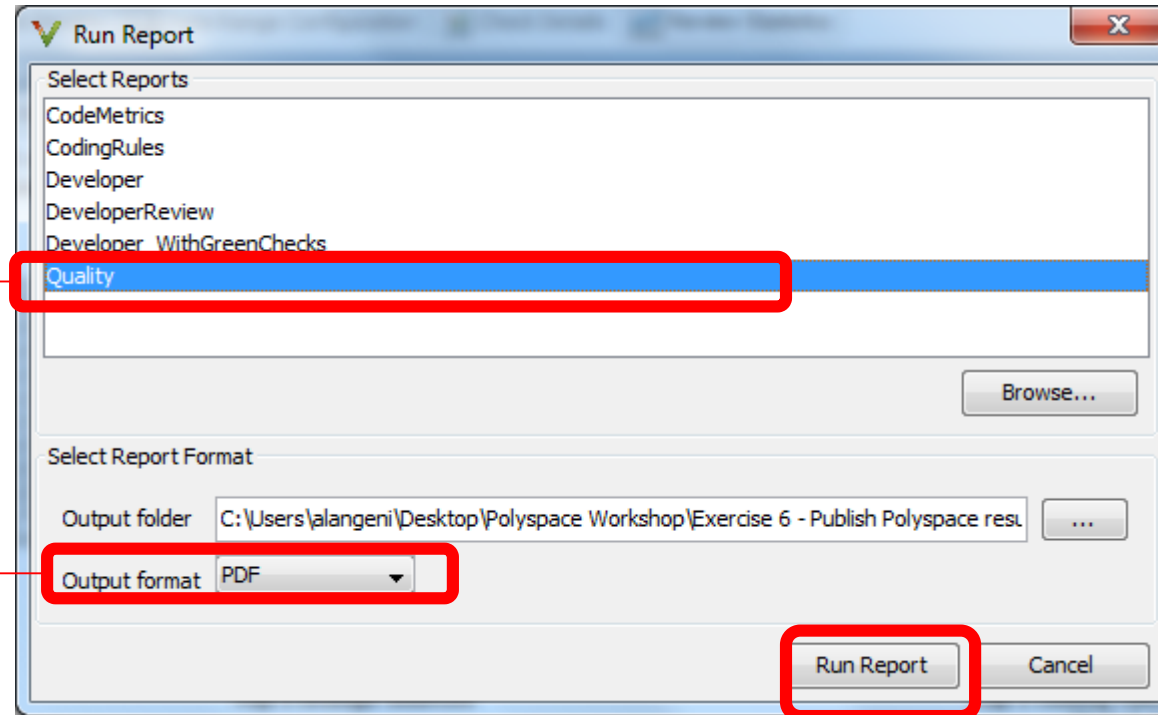


赤点線に対する「原因に移動」

Polyspace結果レポートの生成

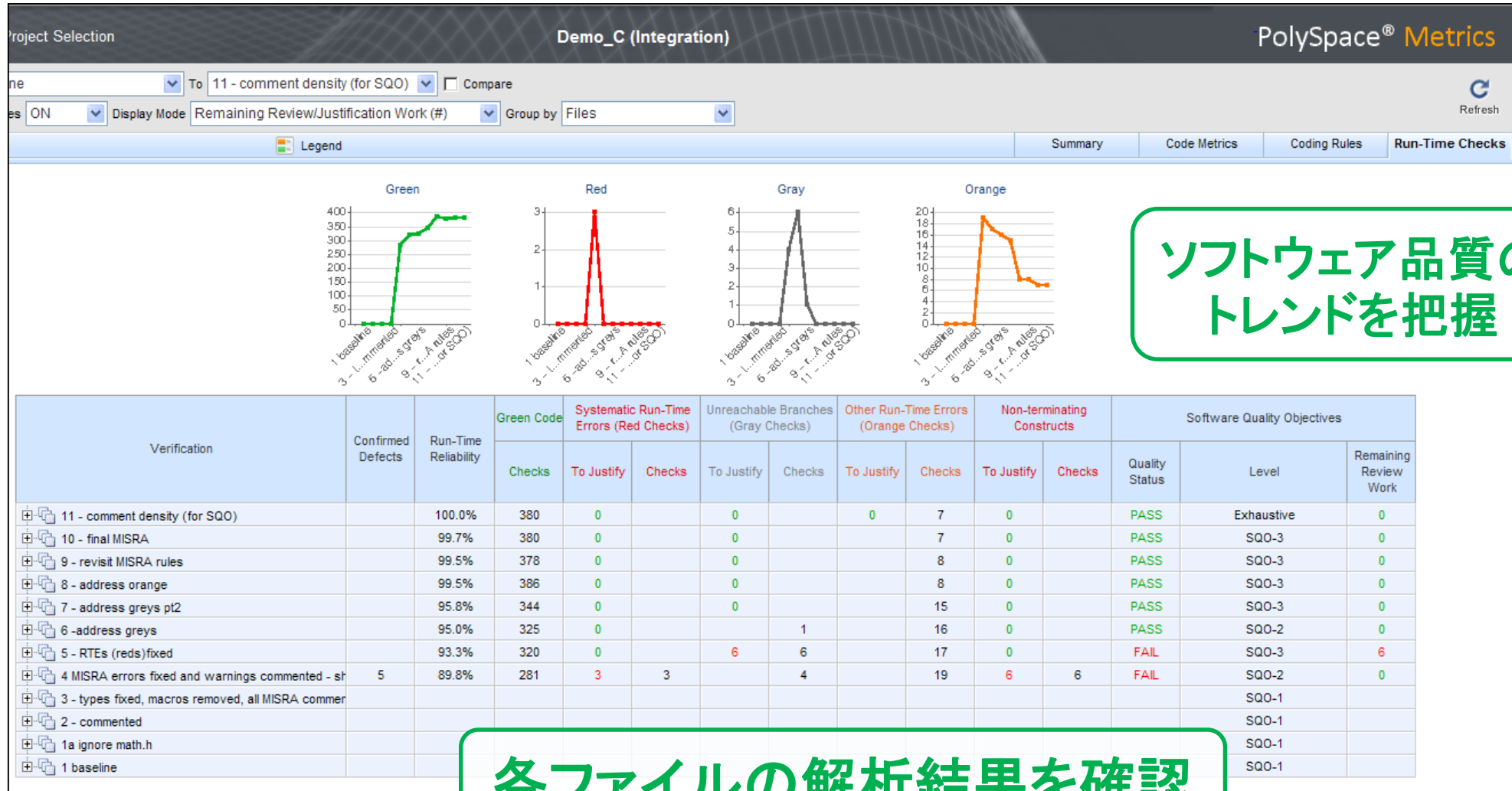
レポートテンプレート

出力形式



レポート生成を実行

Polyspace Metrics: コード品質の管理



アジェンダ

第1部 静的コード解析手法を提供するPolyspace紹介

第2部 Polyspaceを利用した静的コード解析

2.1 Polyspaceプロジェクト作成

2.2 レビュー時のTips & 便利機能



2.3 まとめ

第3部 Q&A会

規格準拠に向けて

- コーディング規約
 - MISRA-C
 - MISRA-C++
 - JSF++
- ソフトウェアメトリクス
 - HIS Source Code Metrics
 - <http://www.automotive-his.de/>
- 認証規格
 - DO-178B
 - DO Qualification Kit
 - IEC 61508, ISO 26262, EN 50128
 - IEC Certification Kit



Polyspace製品で目標規格達成をアシストします！

Applicability to ISO 26262 (Example)


ISO 26262-6 Software unit design and implementation

| Methods | | ASIL | | | | Applicable Tools / Processes |
|---------|--------------------------|------|----|----|----|--|
| | | A | B | C | D | |
| 1a | Walk-through | ++ | + | o | o | Polyspace Bug Finder, Polyspace Code Prover |
| 1b | Inspection | + | ++ | ++ | ++ | |
| 1c | Semi-formal verification | + | + | ++ | ++ | |
| 1d | Formal verification | o | o | + | + | Polyspace Code Prover |
| 1e | Control flow analysis | + | + | ++ | ++ | Polyspace Bug Finder, Polyspace Code Prover |
| 1f | Data flow analysis | + | + | ++ | ++ | |
| 1g | Static code analysis | + | ++ | ++ | ++ | |
| 1h | Semantic code analysis* | + | + | + | + | Polyspace Code Prover |


Table 9 – Methods for the verification of the software unit design and implementation

* ... is used for mathematical analysis of source code by use of an abstract representation of possible values for the variables. For this it is not necessary to translate and execute the source code.
(ISO 26262-6, table 9, Method 1h)

Polyspaceユーザー事例



PCCN
Controls



ONE FORD
ONE TEAM • ONE PLAN • ONE GOAL

Introducing Polyspace into the Software Development Process

Eileen Davidson
Ford Motor Company

12-May-2015

USER STORY

GlucoLight Ensures Reliable Software for Medical Trials Using PolySpace™ Products for C/C++

Studies show that management of glucose levels reduces infection, length of hospital stay, and mortality for patients in intensive care. Current glycemic control methods are time-consuming and labor-intensive, however, requiring medical staff to draw blood samples and manually measure blood glucose levels.

GlucoLight Corporation is developing a noninvasive, continuous glucose monitoring system that uses imaging technology and requires no manual intervention.



GlucoLight SENTRIS-100, a noninvasive glucose monitoring system.

miracor

medical systems

PiCSO Impulse System – From Prototype to Product

Dr. Martin Haidacher
SDMD Europe
Munich

MathWorks®

MATLAB&SIMULINK

日産自動車

Polyspaceを利用してソフトウェア品質を向上

課題

ソフトウェア品質の改善に向け、これまで見つけれなかったランタイムエラーを発見すること

ソリューション

MathWorksのツールを利用して、日産やサプライヤが作成したコードを完璧に解析

結果

- サプライヤのコードにバグを見つけることができた
- ソフトウェアの信頼性が改善された
- Polyspace製品が日産のサプライヤに適用できることが分かった



Nissan Fairlady Z.

『Polyspace製品を利用して、それまで業界のどのツールもできなかったソフトウェアの信頼性を確保することができました。』

日産自動車 菊池様

Polyspace ソースコード静的検証 ～まとめ～

- Polyspace Bug Finderで
素早くコードの欠陥を検出！
早期段階で不具合の修正が可能！
- Polyspace Code Proverで
クリティカルシステムにランタイム
エラーの有無を証明！
- 両ツールを使用して
効率的にソフトウェアの品質を
管理・向上！

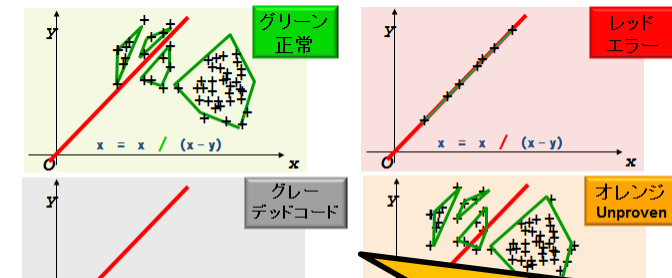
Simulink環境から実行可能
ファイル・プロジェクト単位で使用可能

```

extern void useint(int val);
void bug_partiallywrittenarray(void)
{
    int tab[5] = { 0, 1, 2, 3, 4 };

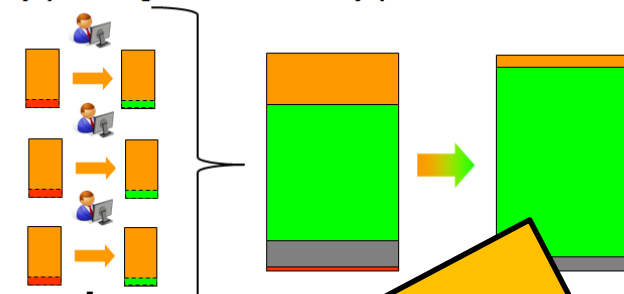
    useint(tab[0]);
    useint(tab[1]);
    useint(tab[2]);
    useint(tab[2]); /* Defect: Same index a
    useint(tab[4]);
}
  
```

時間を掛けずに大部分のバグを識別



形式手法の解析によるコード信頼性

Polyspace Bug Finder → Polyspace Code Prover



安全なコードを確保して実機実験へ

第3部 Q&A会

Polyspace の有効活用に向けて

短期間で習得していただけるような教育カリキュラムをご提供します

PolyspaceによるC/C++コード検証(2日間)

*費用概算 70万円(税抜) 10名様まで

第 1 日

- Polyspace のワークフローの概要
- Polyspace Bug Finder™ 解析
- Polyspace Code Prover™ 検証結果の解析

第 2 日

- Polyspace Code Prover™ 検証と結果の管理
- Polyspace Code Prover™ 検証への精度の追加
- 統合解析
- ソフトウェア品質の測定とレポート
- アプリケーション解析





Accelerating the pace of engineering and science

© 2020 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.