JaSST'19 Tokyo
2019/03/27(水)

C/C++コードに潜むランタイムエラーの流出を阻止する安全性証明

MathWorks Japan
アプリケーションエンジニアリング部（制御）
アプリケーションエンジニア
田中 康博

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 }
```

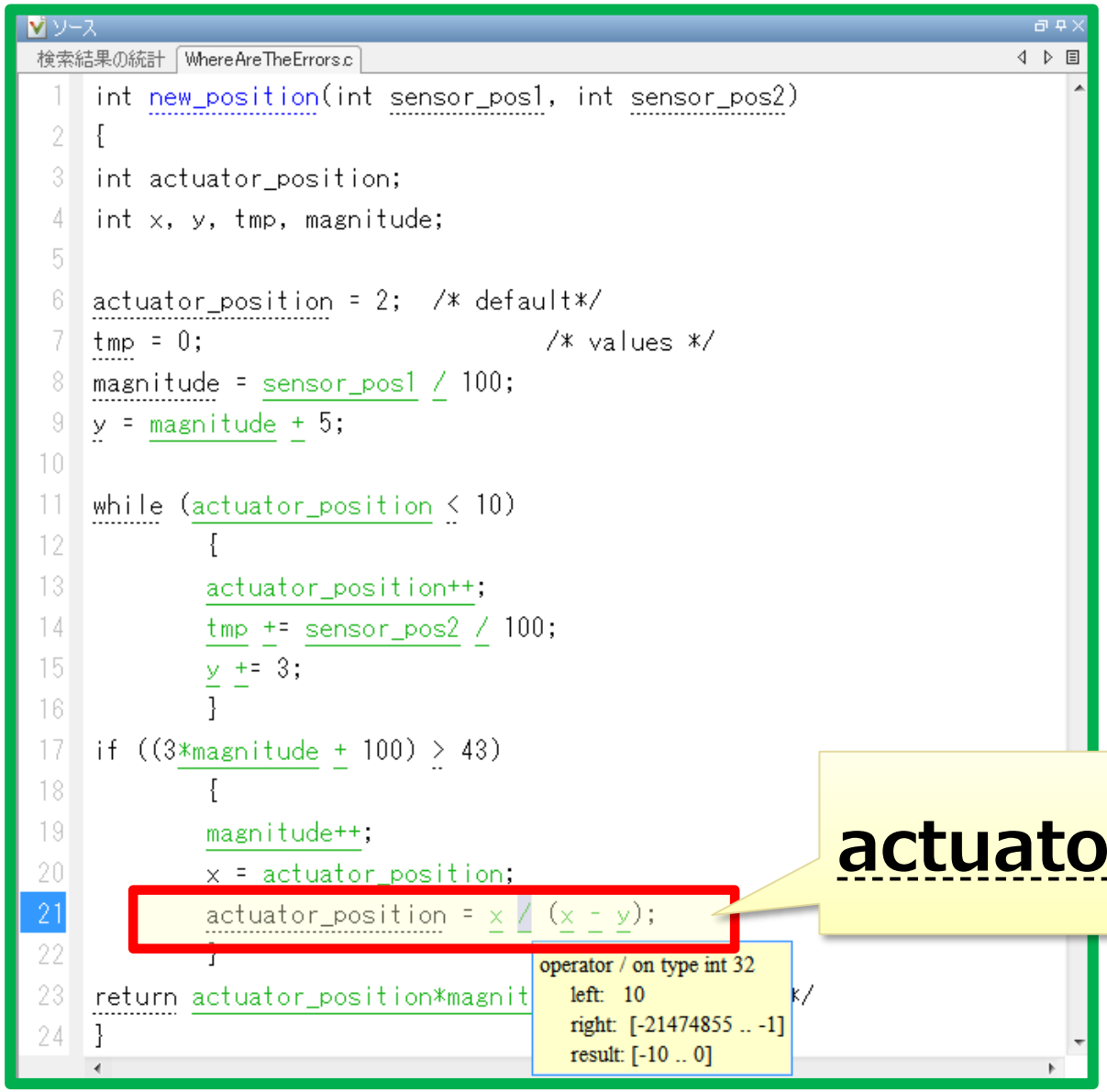
ゼロ除算の可能性

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 によるコードの安全性証明

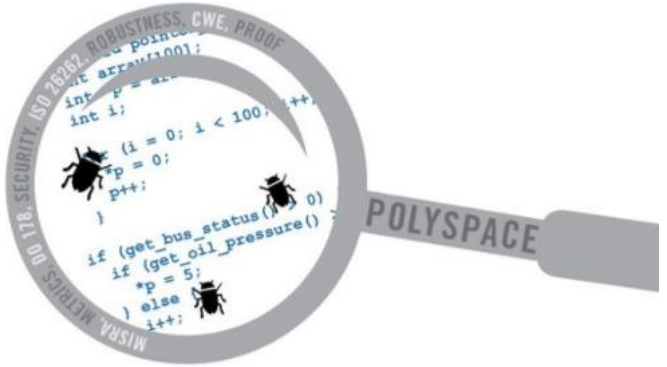


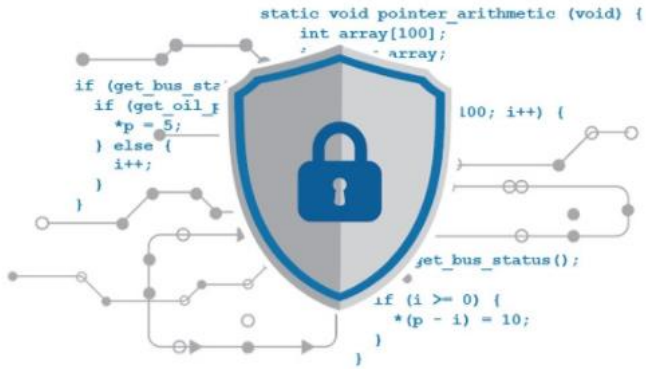
- ✓ あらゆる条件でコードの**安全性**を証明
- ✓ テストを行わずに網羅的に解析
  - **ランタイム(実行時)エラー**
  - **条件によるランタイムエラー**
  - **到達不能なコード**

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

アジェンダ

1. 静的解析とは
2. 抽象解釈とは
3. 抽象解釈の事例紹介
4. Polyspace最新情報
5. ユーザー事例と機能安全対応





アジェンダ

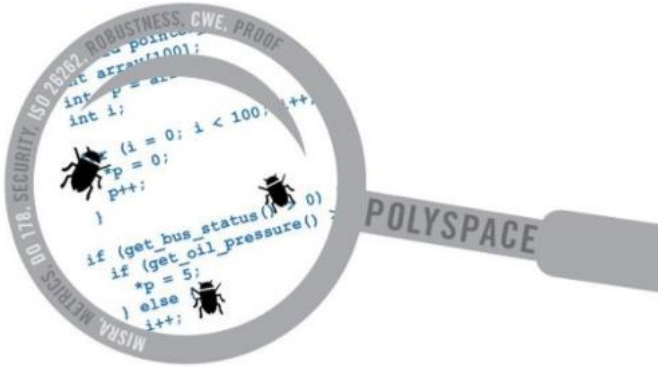
静的解析とは

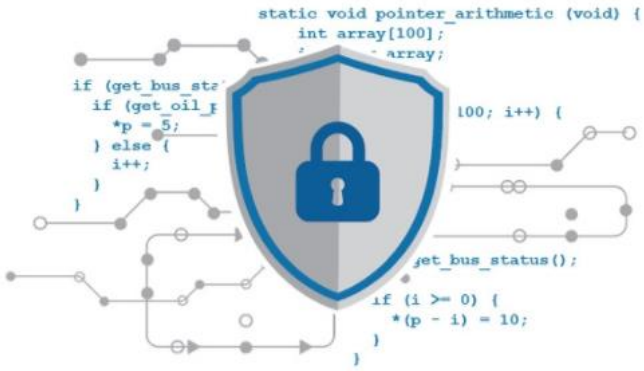
2. 抽象解釈とは

3. 抽象解釈の事例紹介

4. Polyspace 最新情報

5. ユーザー事例と機能安全対応





ソフトウェア品質管理技法

テスト手法を使い分けて効率的にソフトウェアの品質を向上させる

- 動的手法
  - 動的解析
  - テスト
  - シミュレーション
- 静的手法
  - レビュー
  - 静的解析
  - 形式手法

コード静的解析ツール

静的解析：
Polyspace Bug Finder™

形式手法：
Polyspace Code Prover™

ソフトウェア品質管理技法 (SWEBOK v3.0より)

静的解析とは – Polyspace Bug Finder

プログラムを実行せずにソフトウェアの品質と信頼性を検証する

静的解析でできること

- **バグの検出**
  - 未初期化変数、メモリリークなど
- **セキュリティ脆弱性の検出**
  - CERT-C, CWE, ISO17961など
- **コーディング規約のチェック**
  - MISRA-C:2004, 2012など
- **コードメトリクスの計測**
  - 循環的複雑度、コメント密度など

静的解析の特徴

- プログラムを実行せずに検証できる
- テストケースを作成せずに検証できる
- 機械的にバグを検出できる
- 解析速度が速い
- ランタイムエラーの検出が難しい

形式手法で解決可能！

形式手法とは – Polyspace Code Prover

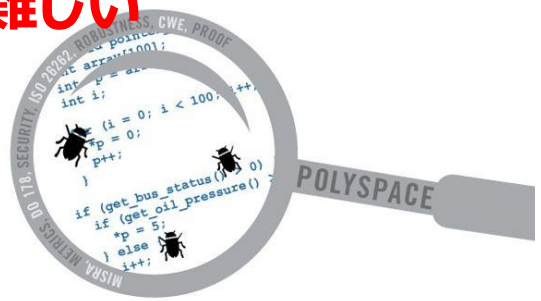
プログラムを数学的に解析してソフトウェアの安全性を証明する

形式手法でできること

- **ランタイムエラーの検出**
  - ゼロ除算、オーバーフロー
  - 配列の範囲外アクセス
  - 到達不能コードなど
- **ランタイムエラーがないことの証明**
- **あらゆる条件を網羅的に検証**

形式手法の特徴

- **プログラムを実行せずに検証できる**
- **テストケースを作成せずに検証できる**
- **数学的な理論に基づいて検証できる**
- **網羅的に解析する/未検出がない**
- **開発者への必要性やノウハウ・知識の浸透が難しい**



静的解析ツールの手法と特徴

Sound Tool vs Unsound Tool

ツールの種類	解析の手法	解析の特徴
Sound Tool	• 形式手法 (抽象解釈)	• コードの安全性を証明 • 未検出なし • 解析時間が長い
Unsound Tool	• 静的解析 - シンタックス解析 - セマンティックス解析 - 制御フロー解析 - データフロー解析	• 誤検出あり/なし • 未検出あり • 解析時間が早い

アジェンダ

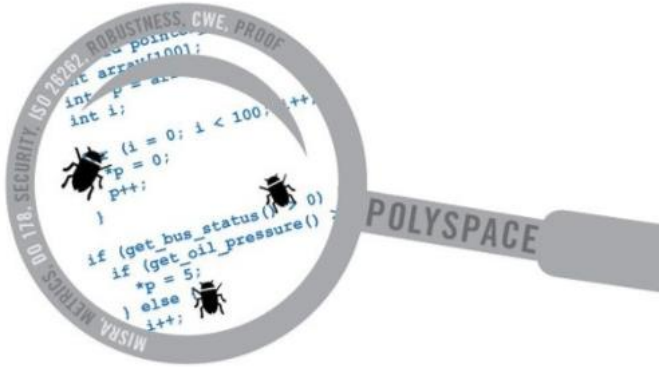
1. 静的解析とは

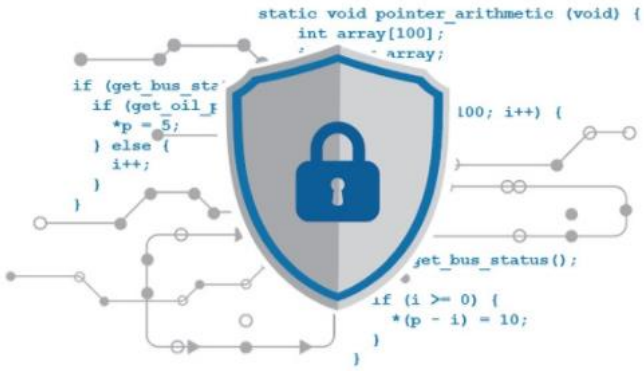
2. 抽象解釈とは

3. 抽象解釈の事例紹介

4. Polyspace 最新情報

5. ユーザー事例と機能安全対応

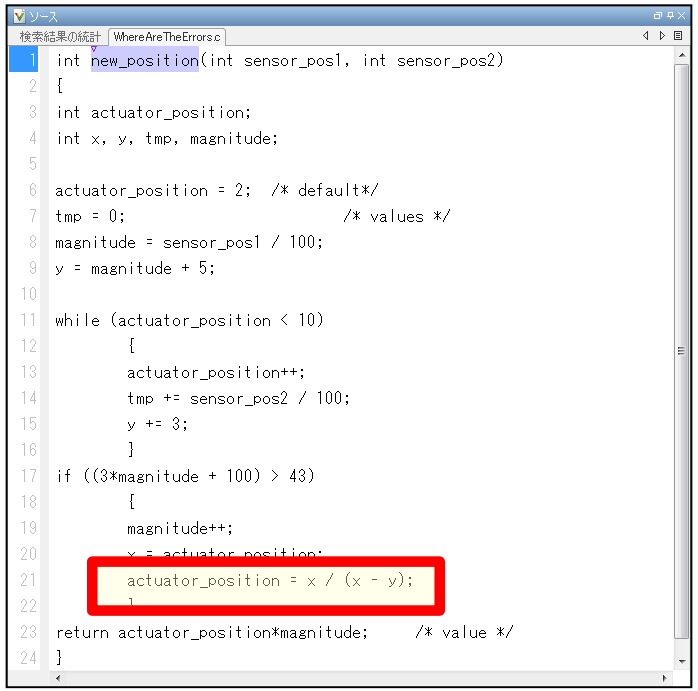




Polyspace の抽象解釈によるコードの安全性証明

ランタイムエラーの可能性は？

```
x = x / (x - y)
```



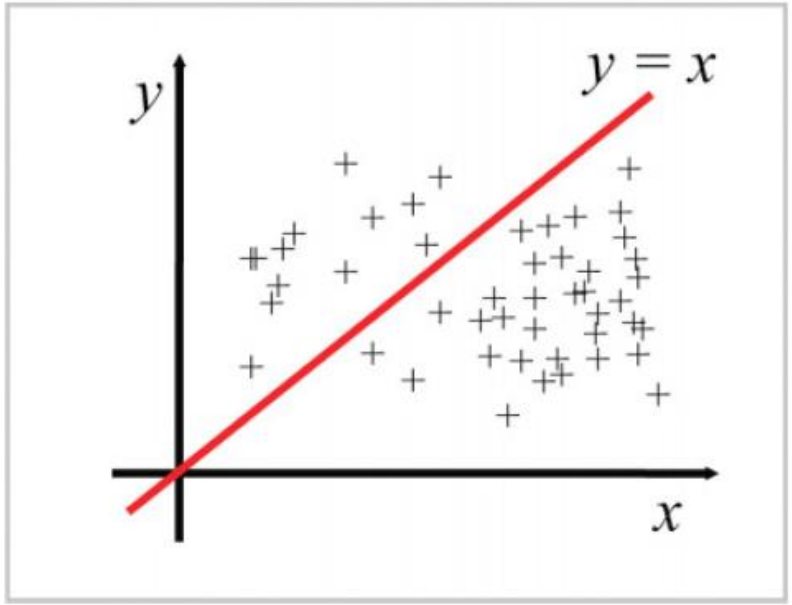
- ゼロ除算の可能性
- オーバーフローの可能性
- 未初期化の可能性

Polyspace の抽象解釈によるコードの安全性証明

理想：全てのテストケースを実行　　現実：時間的に無理

ゼロ除算の検証

```
x = x / (x - y)
```

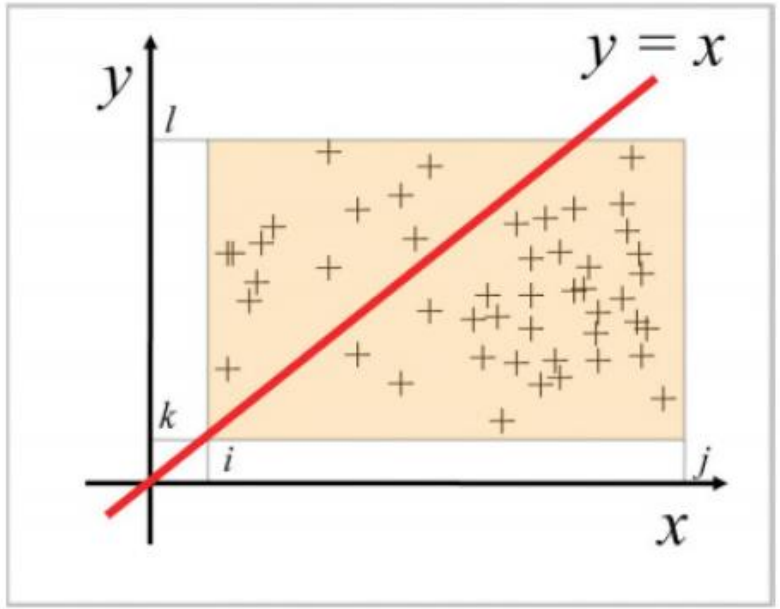


Polyspace の抽象解釈によるコードの安全性証明

タイプ解析の正確度は不十分 → レビューが困難

ゼロ除算の検証

```
x = x / (x - y)
```



タイプ解析
変数レンジ近似

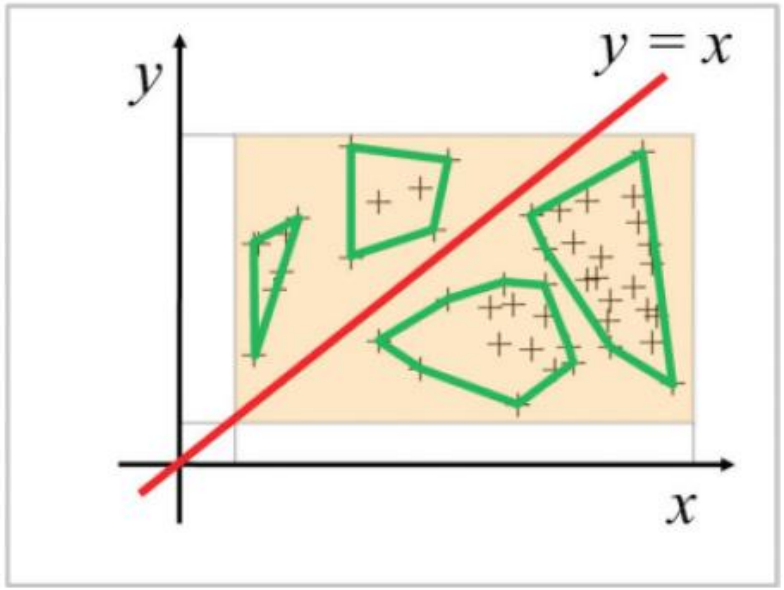
- x の最小値・最大値
- y の最小値・最大値

Polyspace の抽象解釈によるコードの安全性証明

抽象解釈で正確に全てのケースを網羅

ゼロ除算の検証

```
x = x / (x - y)
```



抽象解釈
多角形近似

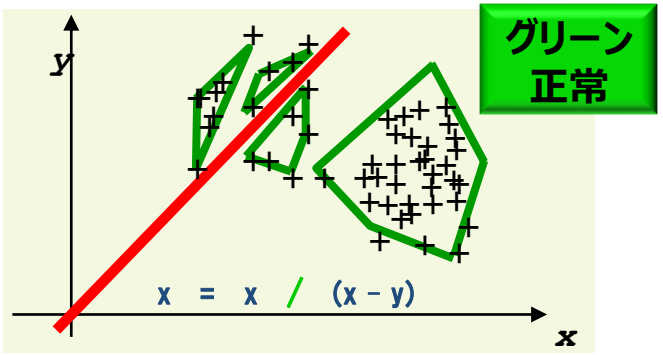
- 制御構造

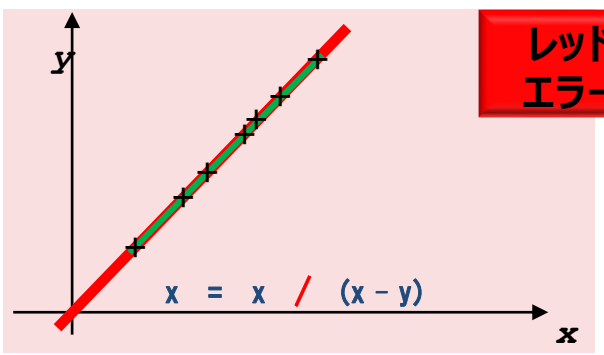
  (if-else, for, while, switchなど)
- 手続き間操作 (関数呼び出し)
- マルチタスク解析

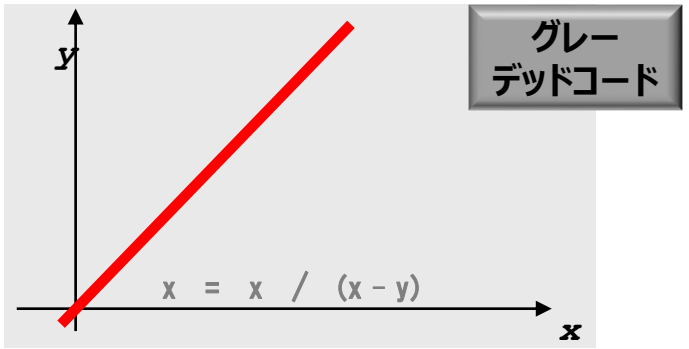
Polyspace による解析結果の分類

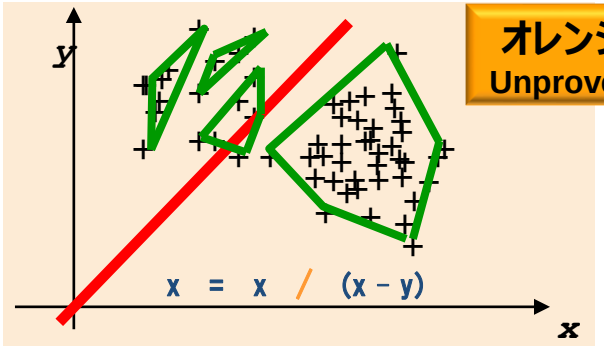
ゼロ除算の検証

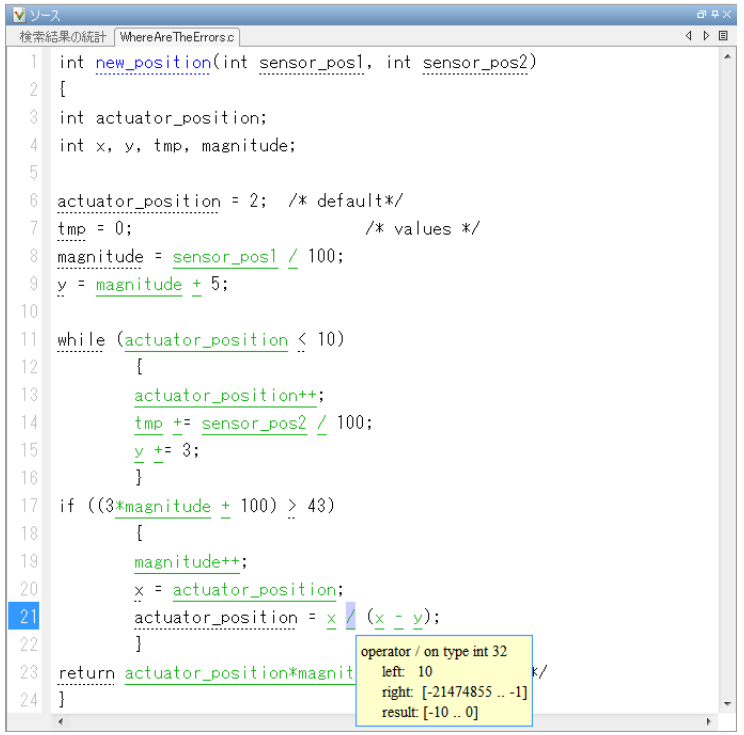
$$x = x / (x - y)$$





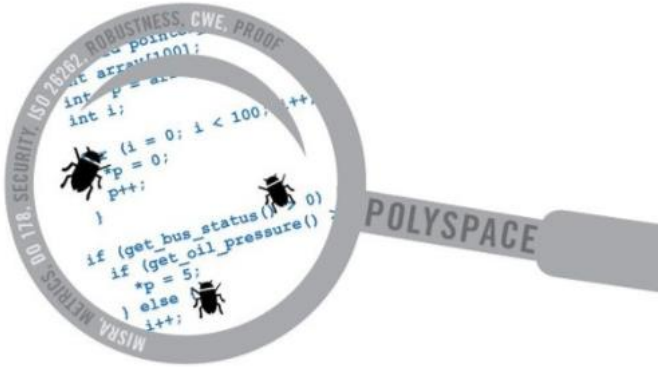


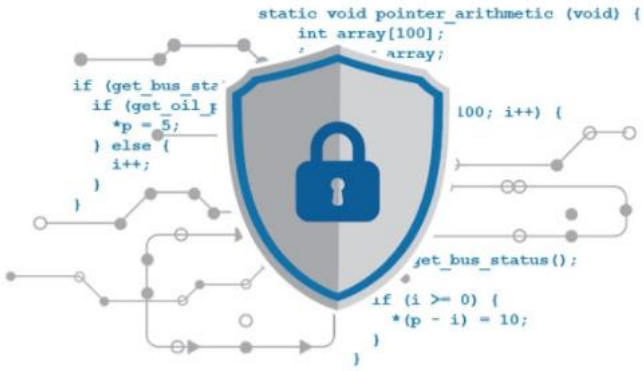




アジェンダ

1. 静的解析とは
2. 抽象解釈とは
3. 抽象解釈の事例紹介
4. Polyspace 最新情報
5. ユーザー事例と機能安全対応





Polyspace による抽象解釈の例

制御構造解析：forループ後の範囲外ポインターアクセス

```
10:  int  ar[100];
11:  int  *p = ar;
12:  int  i;
13:  for (i = 0; i < 100; i++, p++) {
14:    *p = 0;
15:  }
16:  *p = 5;
```

ポインターpを配列arの先頭から始まる

離散増加関数として抽象化

● 不適切にデリファレンスされたポインター

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:
'ar', local to function 'demo1'.

Polyspace による抽象解釈の例

制御構造解析：ネストされたforループによる範囲外配列アクセス

```
20:  int  ar[10];
21:  int  i,j;
22:  for (i = 0; i < 10; i++) {
23:     for (j = 0; j <= 10; j++) {
24:       ar[i - j] = i + j;
25:     }
26:  }
```

? 範囲外の配列インデックス

Warning: array index may be outside bounds : [0..9]
This check may be a path-related issue, which is not dependent on input values
array size: 10
array index value: [-1 .. 0]

	イベント	ファイル	スコープ	行
1	Entering function 'demo2'	__polyspace_main.c	main()	55
2	Iterating on loop: loop entered	demo2.c	demo2()	22
3	Iterating on loop: loop ran 1 time	demo2.c	demo2()	23
4	? Warning: array index may be	demo2.c	demo2()	24

イベント トレースバック表示

Polyspace による抽象解釈の例

手続き間解析：ゼロ除算

```
30: void foo(int* depth) {
31:   float advance;
32:   *depth = *depth + 1;
33:   advance = 1.0/(float)(*depth);
34:   if (*depth < 50)
35:     foo(depth);
36: }
37:
```

? ゼロ除算
Warning: float division by zero may occur
operator / on type float 64
left: 1.0
right: integer values in [-3.0 .. 0.0] or
result: [-1.0 .. -0.49999] or [-0.333333

```
38: void bug_in_recursive() {
39:   int  x;
40:   if (random_int()) {
41:     x = -4;
42:     foo(&x);
43:   } else {
44:     x = 10;
45:     foo(&x);
46:   }
47: }
```

アジェンダ

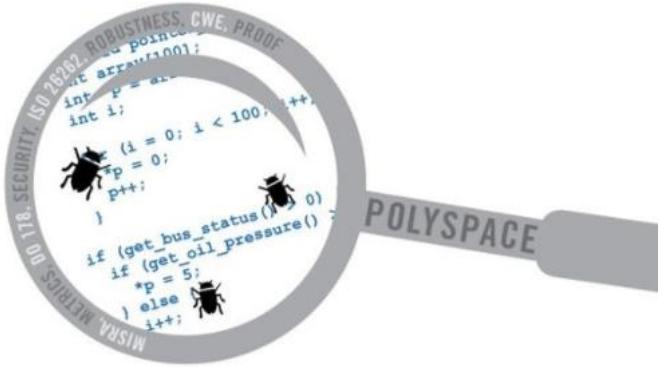
1. 静的解析とは

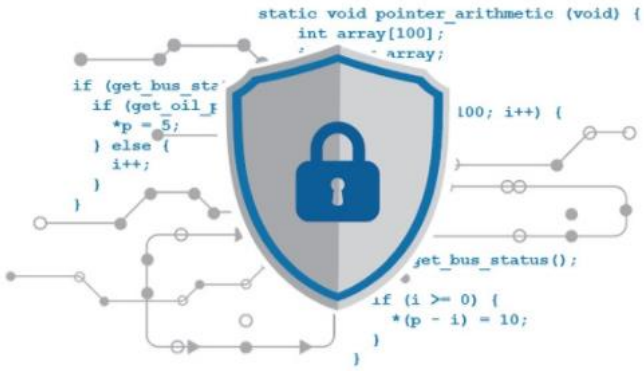
2. 抽象解釈とは

3. 抽象解釈の事例紹介

Polyspace 最新情報

5. ユーザー事例と機能安全対応



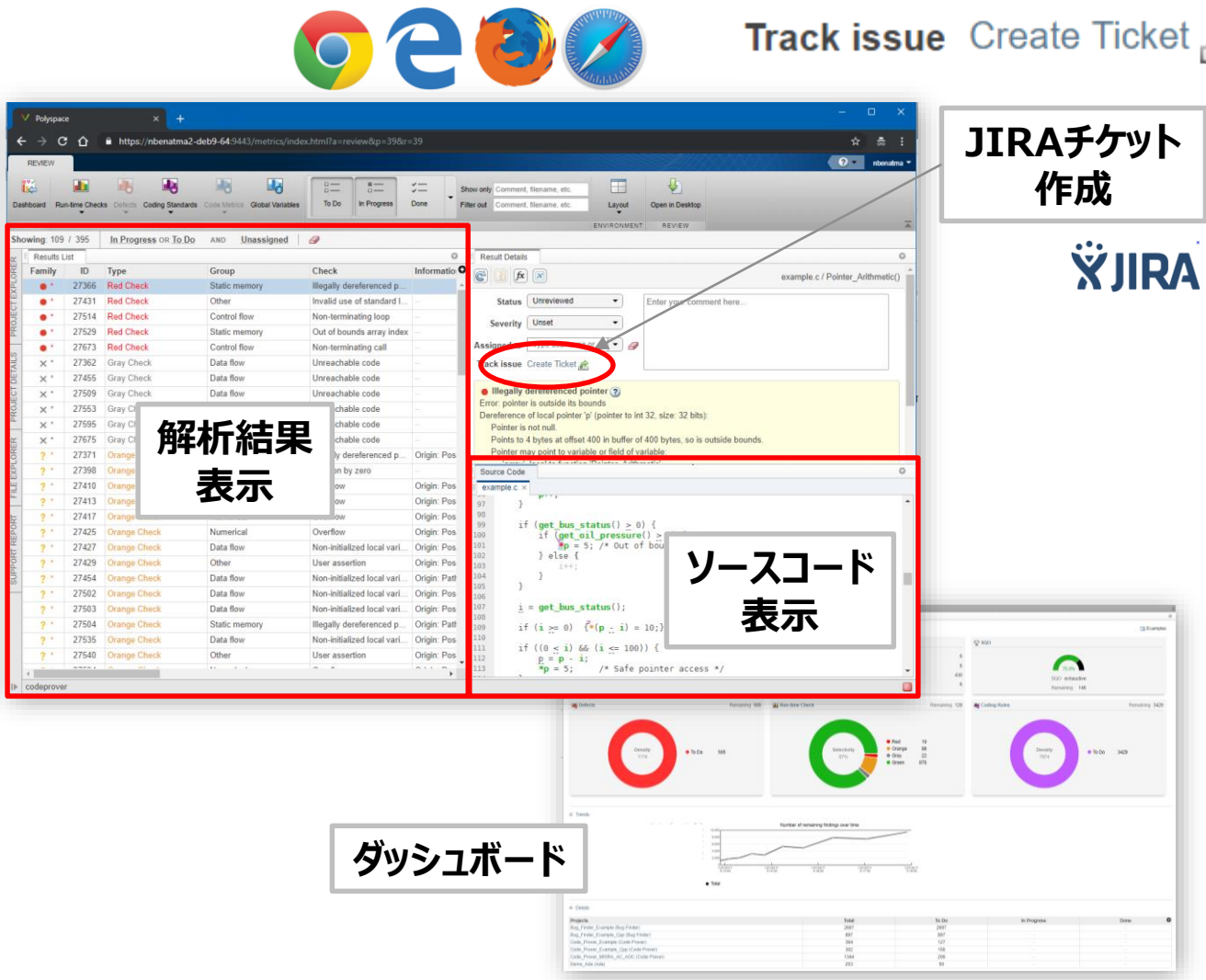


Polyspace Access製品

Polyspace Bug Finder Access and Polyspace Code Prover Access

新製品！

- チームベースのコラボレーション
  - Chrome, Edge, Firefox, Safari
  - 解析結果 / ソースコードをブラウザで表示
  - デバッグ情報の表示
  - 担当者の割り当て, 解析結果の正当化
- ソフトウェア品質の管理
  - ダッシュボードで品質と問題を監視
  - コードメトリクス、SQO
  - バグの傾向追跡
- 規格遵守の追跡
  - 標準への進捗状況を追跡
  - 標準からの逸脱を正当化
  - MISRA, CERT-C など



解析の自動化とチーム間レビュー

[サーバー側]

Web Server
- Code Prover Access
- Bug Finder Access

Polyspace Access製品

接続

解析結果アップロード

自動解析

Jenkins

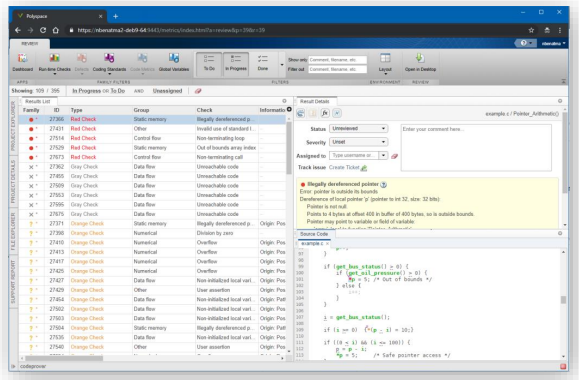
ソースコードリポジトリ

- Code Prover Server
- Bug Finder Server

Polyspace Server製品

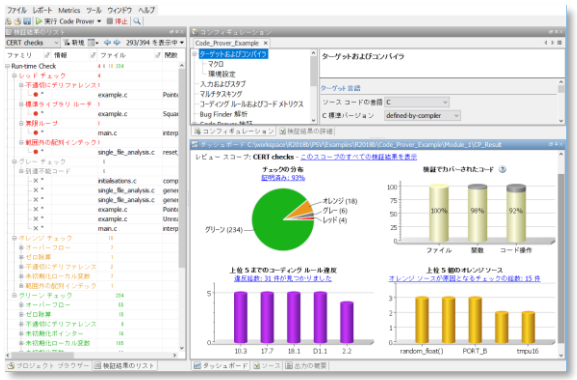
[クライアント側]

解析結果レビュー



起動

デスクトップ解析



マネージャー チームリーダー　品質管理者　レビューワー

開発担当者

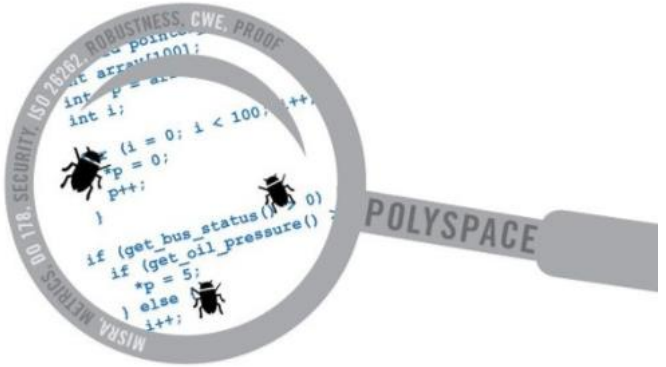
- Code Prover
- Bug Finder

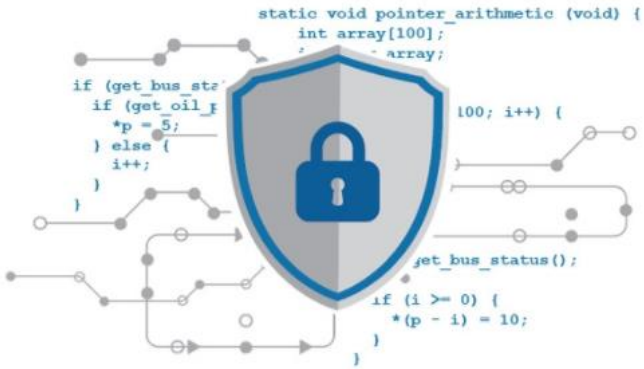
Polyspace Desktop製品

アジェンダ

1. 静的解析とは
2. 抽象解釈とは
3. 抽象解釈の事例紹介
4. Polyspace 最新情報

ユーザー事例と機能安全対応





Polyspace ユーザー事例

医療、航空、自動車など様々な分野で利用されています

日産自動車

ソフトウェアの信頼性向上

TGV高速鉄道

ランタイムエラーの識別加速

アレーニア・アエルマッキ

MISRA-C, DO-178B準拠

Ford

SW開発プロセスに統合

miracor

医療機器の安全性を証明

ソーラー・インパルス

コード証明のよる開発期間の短縮

規格準拠に向けて
Polyspace製品で目標規格達成をアシストします！

- コーディング規約
  - MISRA-C: 2004,2012
  - MISRA-C++: 2008
  - 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

最後に．．．

Polyspaceの評価に

ご興味ありましたらご連絡ください

（yasuhiro.tanaka@mathworks.co.jp）

Polyspaceの詳細に

ご興味ありましたら

展示ブースにお立寄りください

ご清聴ありがとうございました

Accelerating the pace of engineering and science