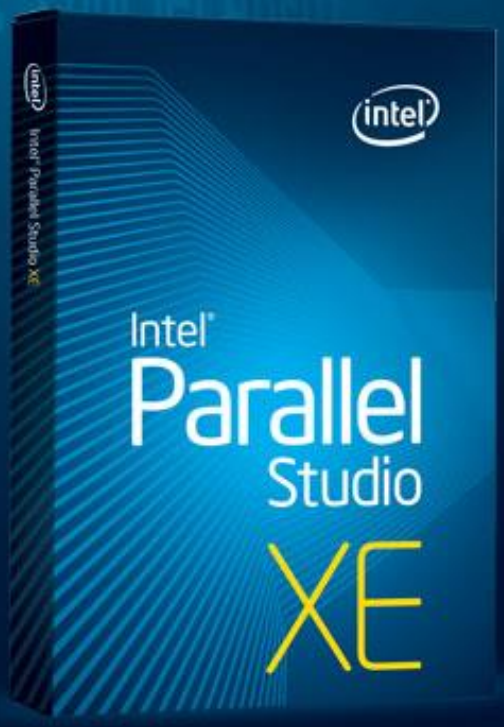
スタティック解析による Fortran コード品質の向上



本ガイドは、スタティック解析により問題を排除し、Fortran コードの安全性を高めて、ソフトウェアの品質を向上する方法について説明します。機能の概要を紹介し、エンドツーエンドの例を用いて使用方法を示します。

このスタティック解析は、インテル® Parallel Studio XE およびインテル® Cluster Studio XE で利用可能です。

スタティック解析による Fortran コード品質の向上

はじめに

本ガイドは、インテル® Parallel Studio XE のスタティック解析機能について説明する入門チュートリアルです。機能の概要を紹介し、エンドツーエンドの例を用いて使用方法を示します。

スタティック解析とは?

スタティック解析は、ソースコードの詳細な解析を通じて、エラーとセキュリティーの脆弱性を発見します。開発者と QA エンジニアが開発サイクルの初期にソフトウェアの問題を検出することで、時間とコストを節約し、投資収益率 (ROI) を改善できるように支援します。また、セキュリティー攻撃に対してアプリケーションを強化する手助けもします。スタティック解析は不具合 (特にテストで完全に発見することが困難な不具合) を発見する効率的な方法を提供します。また、OpenMP* やインテル® Cilk™ Plus などの並列プログラミング・フレームワークの誤用による競合状態も検出します。

コーディング・エラーや安全ではない使用の多くのパターンは、スタティック解析で発見することが可能です。ダイナミック解析と比較して、スタティック解析の主な利点は、テスト中に発生するものだけではなく、可能性のあるすべての実行パスと変数値が検証されることです。通常、セキュリティー攻撃は予期しない方法やテストを実施していない方法で行われるため、この利点は、安全性の保証において特に重要でしょう。

スタティック解析は 250 以上のエラー条件を検出することができます。検出されるエラーには、バッファー・オーバーフロー、境界違反、ポインターとヒープストレージの誤用、メモリーリーク、初期化されていない変数とオブジェクトの使用、C/C++ または Fortran 言語拡張とライブラリーの安全ではない使用/誤用などがあります。

スタティック解析はどのように動作するか

スタティック解析はインテル® C++ コンパイラーおよびインテル® Fortran コンパイラーの特別なモードで実行します。このモードでは、コンパイラーは解析により時間をかけ、命令生成プロセス全体を省略します。これにより、通常のコンパイルでは検知されないエラーを発見することができます。スタティック解析を利用するには、インテル® コンパイラーで重大なエラーなしにコードをコンパイルできる必要がありますが、そのコンパイルの結果として作成されるバイナリーは実行しません。スタティック解析を利用するために、インテル® コンパイラーを使用して製品版のバイナリーを作成する必要はありません。本ガイドでは、スタティック解析を使用するためのアプリケーションの準備方法から説明します。

スタティック解析は、プログラムやライブラリーの一部だけにも使用できますが、プログラム全体に使用することで最も効果を発揮します。各ファイルは 1 つのオブジェクト・モジュールにコンパイルされ、解析結果はリンク時に生成されます。そして、その結果はインテル® Inspector XE を使用して表示されます。

スタティック解析の結果は、多くの場合、決定的ではありません。ツールの結果は、調査に値する、潜在的な問題の最良の考察です。修正が必要かどうかを決定するのは開発者自身です。インテル® Inspector XE GUI は、このプロセスを円滑に行うために設計されています。報告された結果の各問題に対してステート (状態) を割り当てることで、解析結果を開発者が決定します。インテル® Inspector XE はそのステート情報を結果ファイルに保存します。

時間が経つにつれて、ソースも変更され、この解析を再度行うことがあるでしょう。インテル® Inspector XE で初めて新しい解析結果を開くと、以前の結果と新しい結果が自動的に対応します。この対応により、古い結果のステート情報が新しい結果に反映されます。つまり、これで同じ問題について再度調査する必要がなくなります。

修正の必要があると判断した問題については、テストで検出された不具合をレポートする場合と同じように、通常のバグ・トラッキング・システムにレポートしてください。インテル® Inspector XE はバグ・トラッキング・システムではありません。ステート情報は、解析結果の調査の進捗を追跡するものです。結果に対して開発者が何を行うかは、スタティック解析やインテル® Parallel Studio XE の範囲外です。

スタティック解析の設定

設定のプロセスは通常、比較的簡単ですが、状況によっては複雑になることもあります。Microsoft* Windows* で Microsoft* Visual Studio* を使用する場合は、メニュー/ダイアログからスタティック解析用のソリューション構成を自動的に作成できます。詳しい操作方法については、後述のチュートリアルで説明します。

Linux* または Windows* で makefile やコマンドライン・スクリプトを使用してビルドする場合は、スタティック解析用の新しいビルド構成を作成する必要があります。「ビルド構成」とは、特定のコンパイラー・オプションを指定し、中間ファイルを特定のディレクトリーに配置してアプリケーションをビルドするモードを指します。ほとんどのアプリケーションでは少なくとも、デバッグとリリースの 2 つの構成があります。もう 1 つをスタティック解析向けに作成してください。スタティック解析構成は、インテル® コンパイラーでスタティック解析を有効にするオプションを指定してビルドする必要があります。

新しいビルド構成を作成することと、既存の構成を追加オプションでビルドすることは異なりますので、ご注意ください。例えば、デバッグ構成でスタティック解析を有効にするオプションを使ってビルドすることで解析結果を得られます (インテル® コンパイラーでビルドした場合)。これは、最初の製品評価としては非常に良い方法です。しかし、この方法を継続的に行うことは不便でしょう。なぜなら、スタティック解析で生成されるオブジェクト・モジュールは、スタティック解析を実行するたびにデバッグモード・オブジェクト・モジュールを上書きするからです。デバッグ・オブジェクト・モジュールをリリース・オブジェクト・モジュールと分けておきたいのと同様に、スタティック解析オブジェクト・モジュールも分けておくと良いでしょう。スタティック解析を継続的に使用する場合は、このように設定を適切に行ってください。

新しいビルド構成を作成するプロセスは、各アプリケーションによって異なります。本ガイドで使用するサンプル・アプリケーションは、Microsoft* Windows* で Visual Studio* を使用するか、Linux* で makefile を使用してビルドできます。ここでは、この makefile をスタティック解析向けに更新する手順を示します。

アプリケーションのビルドが非常に複雑で、安全に変更できるか分からない場合は、別の設定方法があります。inspxe-inject と呼ばれる「ウォッチャー」ユーティリティーで通常のビルドを実行できます。このアプリケーションはプロセスの作成を中断して、ビルド中に行われたすべてのコンパイルとリンク段階を認識し、この情報をビルド仕様ファイルに記録します。このファイルは、別のユーティリティー inspxe-runsc の入力ファイルとして使用します。inspxe-runsc はインテル® コンパイラーを起動して、オリジナルのビルドと同じビルドステップを繰り返します。これらのユーティリティーについては、ここでは説明していません。

Fortran チュートリアル

インテル® Parallel Studio XE のダウンロードとインストール

(まだインテル® Parallel Studio XE がインストールされていない場合)
推定所要時間: 15-30 分

1. インテル® Parallel Studio XE の評価版をダウンロードします。
2. parallel_studio_xe_2013_setup.exe をクリックしてインテル® Parallel Studio XE をインストールします (システムにより異なりますが、約 15-30 分かかります)。

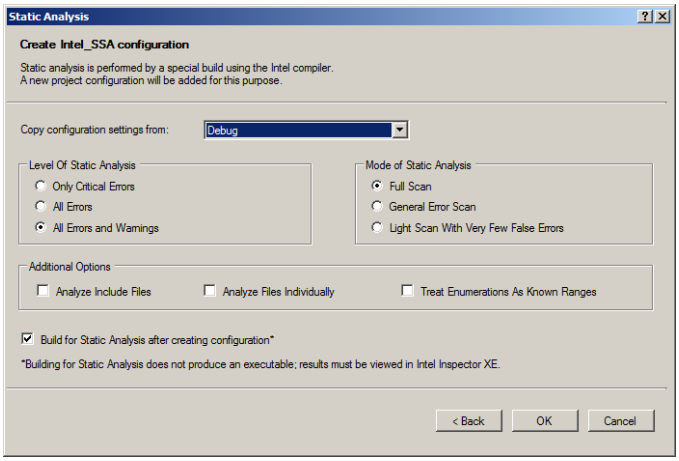
ここでは、「FortranDemo_ssa」というサンプル・アプリケーションを使用します。このサンプルは、インテル® Inspector XE のインストール・ディレクトリー以下の「samples」サブディレクトリーにあります。このサンプル・アプリケーションを展開してください。

まず、スタティック解析の設定から開始します。これは、Windows* と Linux* では操作が異なります。Windows* については、以下を参照してください。Linux* については、「Linux* で makefile を使用してスタティック解析を設定する」へ進んでください。

Windows* で Microsoft* Visual Studio* を使用してスタティック解析を設定する

サンプル・アプリケーションに含まれている Visual Studio* ソリューション FortranDemo.sln を使用して設定を行います。まず、このソリューションを Visual Studio* で開きます。スタティック解析の設定は簡単です。このソリューションは Visual Studio* 2008 形式なので、Visual Studio* 2010 以降を使用している場合は変換する必要があります。指示に従って、変換を行ってください。

スタティック解析の設定を行うには、**[Build (ビルド)] > [Build solution for Intel Static Analysis (インテル・スタティック解析用にソリューションをビルド)]** メニューを選択します。**[Create configuration (構成を作成)]** ダイアログで、必要に応じて、既存のベースライン構成を選択し、スタティック解析オプションを指定します。



[Level of Static Analysis (スタティック解析のレベル)] は、次のいずれかに設定することができます。

a) **[Only Critical Errors (致命的なエラーのみ)]** は、重要度が「critical (クリティカル)」の診断のみをレポートします。重要度が「error (エラー)」または「warning (警告)」の診断はレポートされません。

b) **[All Errors (すべてのエラー)]** は、重要度が「critical (クリティカル)」および「error (エラー)」の診断をすべてレポートします。重要度が「warning (警告)」の診断はレポートされません。

c) **[All Errors and Warnings (すべてのエラーと警告)]** は、重要度に関係なくすべての診断をレポートします。

[Mode of Static Analysis (スタティック解析のモード)] は、次のいずれかに設定することができます。

a) **[Full Scan (フルスキャン)]** モードは、プログラムのすべての脆弱性を検出しますが、誤検出の可能性が高くなります。このモードは、セキュリティーを確保する場合に推奨します。これはデフォルトです。

b) **[General Error Scan (一般エラースキャン)]** モードは、より多くの検出漏れを許容することで誤検出を減らします。このモードは、一般的なエラーを検出する場合に推奨します。

c) **[Light Scan With Very Few False Errors (ライトスキャン (少ない誤検出))]** モードは、誤検出を最小限に抑えますが、検出漏れの可能性が高まります。このモードは、承認テストを行う場合に推奨します。

このチュートリアルでは、デフォルト設定のまま **[OK]** をクリックします。すると、ベースラインの (Debug) 構成からプロパティーがコピーされ、選択したスタティック解析関連のオプションを有効にするように変更された新しい Intel_SSA 構成が作成されます。そして、**[Build for Static Analysis after creating configuration (構成を作成後にスタティック解析用にビルド)]** チェックボックスがオンになっていたため、この構成のビルドが直ちに実行されます (つまり、スタティック解析が実行されます)。

以上でスタティック解析の設定と最初の解析は終了です。

コマンドライン・スクリプトや makefile を使用してアプリケーションのビルドを設定する方法については、次に説明する、Linux* での設定手順のセクションを参照してください。

Linux* で makefile を使用してスタティック解析を設定する

サンプル・アプリケーションには Linux* でアプリケーションをビルドするのに使用できる makefile が含まれています。この makefile はインテル® Fortran コンパイラーを使用してアプリケーションをビルドします。ビルドターゲットは SSA です。SSA ビルドターゲットは、次の変更点を除いて、デバッグ・ビルド・ターゲットのようなものです。

1. 追加オプション「-diag-enable:sc3」が含まれている。これにより、スタティック解析は、重要度に関係なくすべての診断をレポートします。これは、Windows* で **[All Errors and Warnings (すべてのエラーと警告)]** を指定するのと同じです。代わりに、「-diag-enable sc1」または「-diag-enable sc2」を指定することができます。「-diag-enable sc1」は、重要度が「critical (クリティカル)」の診断のみをレポートします。重要度が「error (エラー)」または「warning (警告)」の診断はレポートされません。「-diag-enable sc2」は、重要度が「critical (クリティカル)」および「error (エラー)」の診断をすべてレポートします。重要度が

「warning (警告)」の診断はレポートされません。

2. 中間ファイルを SSA サブディレクトリーに置く。

また、-diag-enable sc-*mode* をコマンドラインに追加して解析モードを指定することもできます。*mode* は、full、concise、precise のいずれかです。

1. 「full」モードは、プログラムのすべての脆弱性を検出しますが、誤検出の可能性が高くなります。このモードは、セキュリティーを確保する場合に推奨します。これは、-diag-enable sc{[1|2|3]} が指定されている場合のデフォルトです。
2. 「concise」モードは、より多くの検出漏れを許容することで誤検出を減らします。このモードは、一般的なエラーを検出する場合に推奨します。
3. 「precise」モードは、誤検出を最小限に抑えますが、検出漏れの可能性が高まります。このモードは、承認テストを行う場合に推奨します。

これは、スタティック解析用に必要な makefile ファイルへの変更点を示しています。

設定が終わったので、あとはスタティック解析をビルド構成をビルドし、解析を行うだけです。次の操作を行います。

1. コマンドシェルを開きます。
2. インテル® コンパイラーの bin ディレクトリー以下にある `ifortvars.sh` スクリプトを ia32 オプションを付けて実行し、インテル® コンパイラーの環境変数を設定します。
3. `make ssa` を実行します。

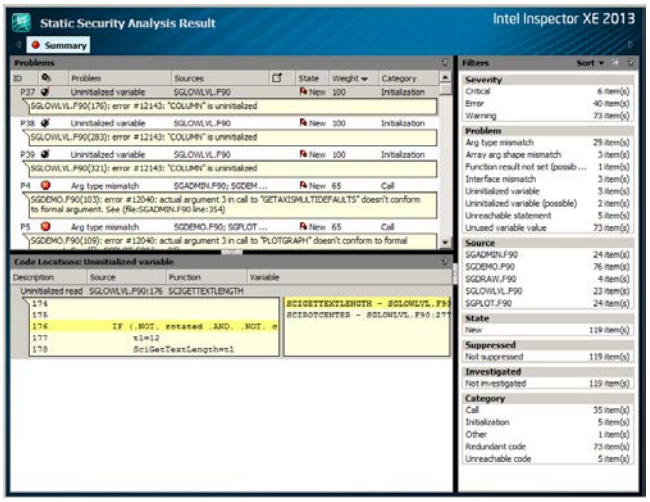
ヒント: 後の操作で使用できるように、このコマンドウィンドウを開いたままにしておきます。

解析結果の確認

Windows* ではビルドが終了すると、新しい結果がインテル® Inspector XE で自動的に開きます。Linux* では `inspxe-gui` と入力してインテル® Inspector XE を起動し、**[File (ファイル)] > [Open (開く)]** メニューから結果を開きます。デフォルトでは、このファイルは「`r000sc.inspxe`」という名前になります。これは、FortranDemo プロジェクトのルート・ディレクトリー以下の `r000sc` ディレクトリーにあります。

このチュートリアルの残りの内容は、Windows* と Linux* でほとんど同じです。主な違いは、Windows* ではインテル® Inspector XE GUI が Visual Studio* に統合されますが、Linux* の GUI は、スタンドアロンのプログラムとして実行されます。インテル® Inspector XE の各ウィンドウの外観も Windows* と Linux* でほとんど同じです。

最初に表示されるウィンドウは次のようになります (注: 必要に応じて、スクロールしてください)。



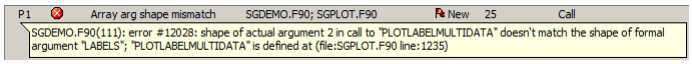
このウィンドウは、3 つの主なエリアで構成されています。左上のペインは **[Problem Sets (問題セット)]** の表です。これが「やること」リスト、つまり調査が必要なものです。左下のペインは、現在選択されている問題セットに対応するコードの場所を示しています。右のペインはフィルターを示しています。どの問題セットを表示/非表示にするかをこのフィルターで制御します。

問題セットの表は、列ヘッダーをクリックして、ソートすることができます。デフォルトでは、**[weight (ウェイト)]** 順にソートされます。ウェイトとは 1 から 100 までの値で、問題のインパクトの程度を表しています。ダメージが大きい可能性があるほど、大きなウェイトの値になります。また、(誤診ではなく) 実際に問題になりそうなものほど高い値になります。つまり、ウェイトは検証する順序のためのガイダンスを提供します。

P1 の Array arg shape mismatch (配列引数の形状の不一致) の問題から何が得られるか、見てみましょう。

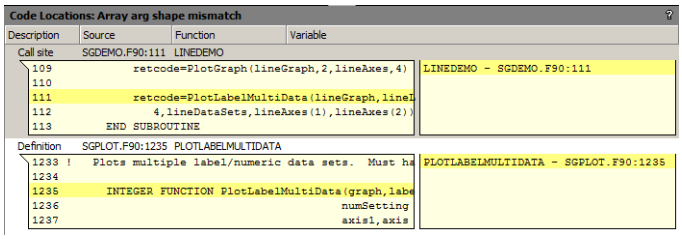
問題の検証

最初に、この問題について分かっていることは、表エントリーにまとめられています。



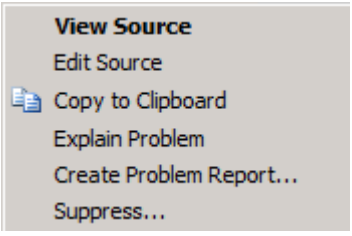
「Array arg shape mismatch (配列引数の形状の不一致)」がこの問題の内容で、説明はその下のシェードエリアに表示されています。ステート列の「New」エントリーは、この問題がこの解析で初めて発見され、まだ調査されていなことを示しています。25 がウェイトの値です。問題のカテゴリーは Call です。これは、実配列引数の異なるサイズの仮引数への割り当てに関連していることを意味します。

この問題をクリックして選択します。下のペインは更新され、この問題に関連するコードの場所が表示されます。



上記のように、2 つのソース個所が表示されます。1 つ目は、関数 PLOTLABELMULTIDATA がサブルーチン LINEDEMO (「Call site」) に呼び出される個所、もう 1 つは、PLOTLABELMULTIDATA が定義される個所です。

問題に関してより多くの情報を得る 1 つの方法は、問題タイプが何かをチェックすることです。いくつかのスタティック解析エラーは非常にテクニカルで、詳細情報が必要です。この問題タイプの詳細情報を見るには、問題を右クリックして、次のようなポップアップ・メニューをが表示します。



[Explain Problem (問題の説明)] を選択して、この問題の詳細が説明されているヘルプトピックを開きます。

Array parameter shape mismatch

The type of an actual argument does not match the corresponding formal parameter at a subroutine call.

Specifically, this error occurs when both the actual and formal parameter types are arrays, but the arrays have different shape.

This same kind of error can also happen when a FORTRAN dummy argument of type subroutine is invoked. That is, the subroutine that is invoked through a dummy argument might exhibit the same problem as can occur in a direct call. In this case, the problem may or may not occur, depending on what subroutine was passed to the dummy argument of subroutine type. There will be an additional observation in such cases that identifies the call site where the subroutine argument was passed in.

ID	Observation	Description
1	Call site	The actual argument that was passed
2	Definition	The definition of the called procedure and shows the declaration of the formal parameter

Example

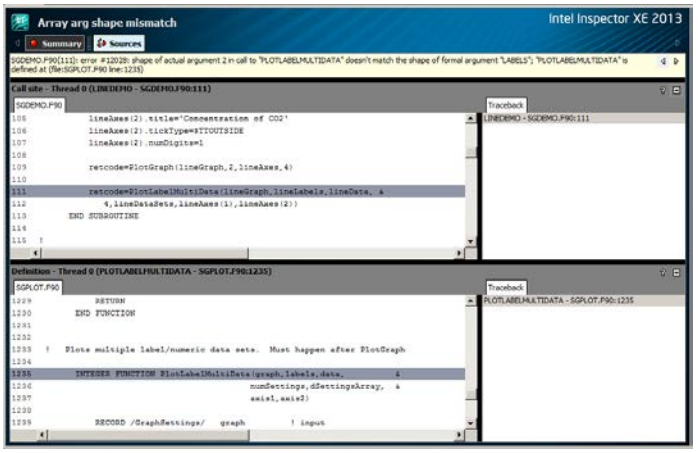
```
subroutine mysub(m)
type mytype1
    integer f1
    real f2
end type mytype1
type (mytype1), dimension(2,3) :: m
print *, m
end

type mytype2
    integer g1
    real g2
end type mytype2
type (mytype2), dimension(3,2) :: n
n%g1 = 1
call mysub(n)
! shapes of dummy argument and actual argument are different.
print *, n%g1
end
```

Copyright © 2010, Intel Corporation. All rights reserved.

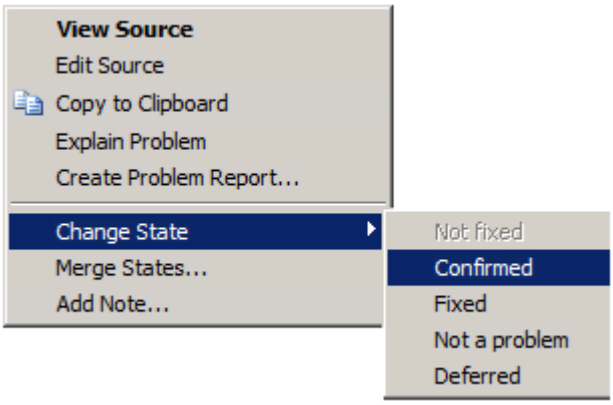
このように、問題タイプのリファレンスでは正確なエラー状態と潜在的な結果について詳しく説明しています。また、問題を示すサンプルも提供しています。

次に、この問題が本当にアプリケーションに存在しているかどうかを判断します。そのためには、ソースコードを確認します。このサンプルでは、コード片からは、問題が何かがよく分かりません。最初のスニペットをダブルクリックして **[Sources (ソース)]** ビューを開きます。

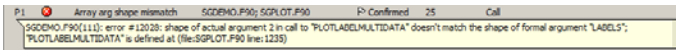


下のソースウィンドウで、PlotLabelMultiData の 2 つ目の引数 labels が 1 つの要素配列であることが分かります。ソースウィンドウで上へスクロールすると、実引数 lineLabels が 6 つの要素を持つ配列として定義されていることが分かります。そのため、これはエラーです。

この問題は (誤診ではなく) 本当のエラーであることが確認できたので、結論を記録しましょう。問題を右クリックし、ポップアップ・メニューから **[Change State (ステートを変更)] > [Confirmed (確認済み)]** を選択します。



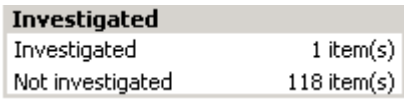
問題セットの表のステートが更新されます。



フィルターで整理

フィルターを使って、開発者は関心のある問題に注目し、無視する問題は非表示にできます。

いったん問題が調査されたら、通常は、それを再度確認する必要はありません。調査が終了したすべての問題を非表示にできるのがフィルターの良い点です。フィルターウィンドウの下部にある **[Investigated (調査済み)]** フィルターの **[Not investigated (未調査)]** をクリックします。



すると、フィルター項目が更新され、アクティブであることが示されます。

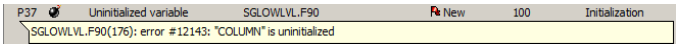
ここで、**[Confirmed (確認済み)]** とマークした問題が、問題セットの表からなくなりました。解析結果の調査中は、このようにフィルターをセットしておくと良いでしょう。

最初の Severity、Problem、Source、State、Category フィルターは問題セットの表の列と対応しています。列の特定の値と一致しない表の行はすべて非表示にすることができます。Source フィルター の 2 行目 (SGDEMO.F90) をクリックすると、次のように表示されます。

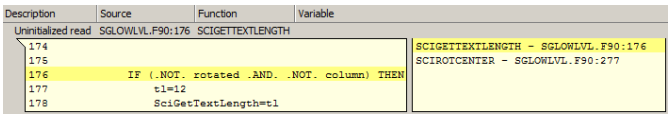
問題セットの表も再描画され、このソースファイルの問題のみが表示されます。**[All (すべて)]** ボックスをクリックして、フィルターをオフにすることもできますが、ここではオンのままにしておきます。

2 つ目の問題の調査

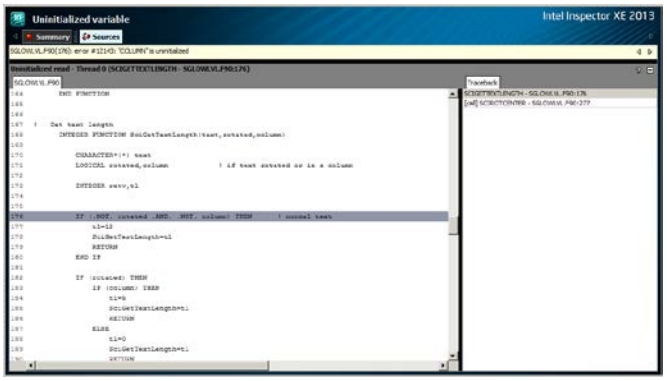
次に、ソリューションの別の問題を見てみましょう。今度は、問題セットの表で「critical (クリティカル)」とマークされている P37 を選びます。



この問題セットを選択して、下のペインで対応するソースコードを確認してみます。



ここでは、column の使用は示していますが、問題が何かはよく分かりません。実際のソースを表示するには、コード片を右クリックし、**[View Source (ソースの表示)]** を選択します。以下のような画面が表示されます。



これが **[Sources (ソース)]** ビューです。1 ペアのウィンドウがあり、ソースコードのセクションと、右側に **[Traceback (トレースバック)]** が表示されます。左上に **[Sources (ソース)]** ボックスがあり、その左に **[Summary (サマリー)]** ボックスがあります。**[Summary (サマリー)]** ボックスをクリックすると、**[Summary (サマリー)]** ビューに戻ります。

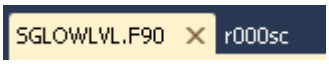
ソースでは、「column」が関数 SciGetTextLength への入力引数であることを示しています。では、SciGetTextLength はどこで呼び出されているのでしょうか? ここで、スタティック解析はプログラム全体のクロスファイル解析を行うことを思い出してください。スタティック解析は、ファイル間をまたがっていても、プロシージャー・コールを通じたデータ値の流れを解析できます。トレースバック情報でこの疑問の答えが分かります。

[Traceback (トレースバック)] を見ると、SciGetTextLength が SciRotCenter に呼ばれていることが分かります。**[Traceback (トレースバック)]** ウィンドウの SCIROTCENTER をクリックすると、ソース・コード・ウィンドウで SciGetTextLength への呼び出しが確認できます。ソース・コード・ウィンドウを上にスクロールすると、SciGetTextLength の呼び出しの前に 'column' が初期化されていないことが分かります。

問題を修正してリスキャン

ソース行 276 に「column = .true.」と入力して、この問題を修正しましょう。**[Sources (ソース)]** ビューの行をダブルクリックするか、行を右クリックしてポップアップ・メニューから **[Edit Source (ソースの編集)]** を選択し、通常のソースエディターを開くことができます。Linux* では、EDITOR 環境変数で指定されているエディターが開きます。Windows* では Visual Studio* ソースエディターが開きます。

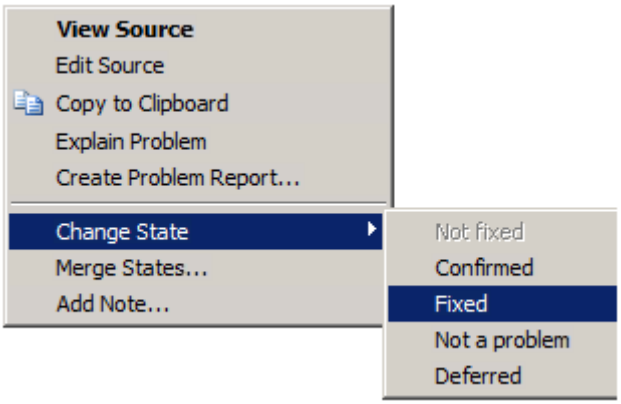
Windows* では、エディターウィンドウは、結果が含まれる同じタブ・グループ・ウィンドウで新しいタブに開きます。ソースを編集用に開くと、結果は隠れます。結果を再度表示するには、**r000sc** タブをクリックします。



編集後、結果を保存してエディターを閉じます。ただし、まだアプリケーションはリビルドしないでください。

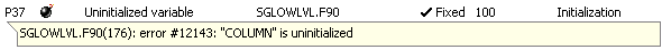
ここで、**[Summary (サマリー)]** ビューに戻りましょう。Visual Studio* で **r000sc** タブをクリックし、インテル® Inspector XE GUI へ戻ります。**[Sources (ソース)]** ビューから **[Summary (サマリー)]** ビューへは、**[Sources (ソース)]** ビューの左上にある **[Summary (サマリー)]** ボックスをクリックします。

ポップアップ・メニューを右クリックして P9 のステートを **[Fixed (修正済み)]** に変更します。



[Fixed (修正済み)] に変更するとすぐに、この問題は表示されなくなります。これは、未調査の問題のみを表示するようにフィルターを設定しているためです。この問題を再度表示するには、**[Filters (フィルター)]** ペインの下部で **[All (すべて)]** をクリックしてフィルターを変更してください。

すると、再度この問題が表示され、**[Fixed (修正済み)]** とマークされていることが確認できます。



ここで、アプリケーションをリビルドし、新しい解析結果、r001sc を生成して開いてみましょう(P37、P38、P39 はすべて関連している問題だったので、リスキャンの前にこの 3 つをすべて調査するほうが一般的ですが、ここでは先にリスキャンします)。Windows* では、**[Build (ビルド)] > [Rebuild solution for Intel Static Analysis (インテル・スタティック解析用にソリューションをリビルド)]** を選択するだけです。ビルドが終了したら、Visual Studio* 出力ウィンドウを閉じてかまいません。Linux* では、前に行ったビルド手順を繰り返します (コマンドウィンドウで makefile を再実行し、インテル® Inspector XE の GUI で **[File (ファイル)] > [Open (開く)]** を選択して r001sc を開きます)。

ここで、ほとんどの問題が「New」ではなく、「Not fixed」となっていることに気付くでしょう。これは、インテル® Inspector XE により、以前の結果 r000sc のステートが新しい結果 r001sc に自動的に反映されるためです。いったん表示された問題は「New」ではなくなり、まだ調査はされていないので、「Not fixed」ステートになります。

また、前に調査した問題 （配列引数の形状の不一致問題） が r000sc でマークしたように、「Confirmed」ステートであることに気付くでしょう。

エラーの数は 6 から 5 に減りましたが、減った問題は、前に指摘されたエラーに関する 1 つである P38 です。r000sc で **[Fixed (修正済み)]** とマークされた P37 が、ここで **[Regression (リグレッション)]** とマークされています。これは、問題がまだ残っていることを示しています。**[Regression (リグレッション)]** は未調査のステートと見なされるため、フィルターを **[Not investigated (未調査)]** ステートのみの表示に設定すると、この問題は表示されます。P38 (r000sc の P39) を調査すると、「column」が初期化されないまま SciGetTextLength の別のインスタンスが呼び出されていることに気付くでしょう。この問題を修正すると、P37 も解決されます。

サマリーとレビュー

これまで行ったことを振り返ってみましょう。

1) 問題セットの表は、スタティック解析で見つかった問題をまとめています。
2) 問題セットを選択すると、関連するコードの場所が下のペインに表示されます。
3) コード片を表示したり、(トラックバックを確認できる) [Sources (ソース)] ビューを開いたり、ソースエディターを開いたりして、問題を掘り下げます。
4) **[Explain Problem (問題の説明)]** メニューで問題タイプのリファレンスを表示し、問題タイプの意味について詳細説明を確認します。
5) 問題のステートを設定して、調査の結果を記録します。
6) フィルターを使用して、調査の終わった問題を非表示にしたり、特定のソースファイルや問題のクラスに注目します。
7) スタティック解析はステート情報を 1 つの結果から別の結果へと自動的に維持します。新規の問題、調査され修正が確認された問題も記録します。

関連情報

ラーニングラボ - テクニカルビデオ、ホワイトペーパー、Webinar など

インテル® Parallel Studio XE 製品ページ - HOW TO ビデオ、入門ガイド、ドキュメント、製品の詳細情報、サポートなど

評価ガイド - さまざまな機能の使用法を紹介する評価ガイド

インテル® ソフトウェア・ネットワーク・フォーラム - デベロッパー・コミュニティー

インテル® ソフトウェア製品ナレッジベース - 製品およびライセンスに関する情報

30 日間の評価版のダウンロード

開発者向け日本語最新技術情報: http://www.isus.jp/

スタティック解析による Fortran コード品質の向上

購入方法: 言語別のスイート

アプリケーションをビルド、検証、チューニングする複数のツールが組み合わされた次のスイートがご利用になれます。本資料で説明している製品は青でハイライトされています。ライセンスは、シングルユーザー・ライセンス、フローティング・ライセンス、アカデミック・ライセンスが用意されています。

	スイート>>	インテル® Cluster Studio XE	インテル® Parallel Studio XE	インテル® C++ Studio XE	インテル® Fortran Studio XE	インテル® Composer XE	インテル® C++ Composer XE	インテル® Fortran Composer XE
コンポーネント	インテル® C/C++ コンパイラー	●	●	●		●	●	
	インテル® Fortran コンパイラー	●	●		●	●		●
	インテル® IPP	●	●	●		●	●	
	インテル® MKL	●	●	●	●	●	●	●
	インテル® Cilk™ Plus	●	●	●		●	●	
	インテル® TBB	●	●	●		●	●	
	インテル® Inspector XE	●	●	●	●			
	インテル® VTune™ Amplifier XE	●	●	●	●			
	インテル® Advisor XE	●	●	●	●			
	スタティック解析	●	●	●	●			
	インテル® MPI ライブラリー	●						
	インテル® Trace Analyzer & Collector	●						
	Rogue Wave IMSL* ライブラリー[2]							●
	オペレーティング・システム [1]	W、L	W、L	W、L	W、L	W、L	W、L、O	W、L、O

注: [1] オペレーティング・システム:W = Windows*、L = Linux*、O = OS X*
[2] インテル® Visual Fortran Composer XE Windows* 版 IMSL* 同梱で利用可能

インテル® Parallel Studio XE の詳細:

- 以下の Web サイトをご覧ください。 http://intel.ly/parallel-studio-xe
- あるいは、左の QR コードをスキャンしてください。

30 日間の評価版:

- http://intel.ly/sw-tools-eval の Web サイトで「Product Suites」をクリックしてください。

著作権、商標、注意事項について

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスを許諾するものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品適格性、あらゆる特許権、著作権、その他知的財産権の非侵害性への保証を含む) に関してもいかなる責任も負いません。

最適化に関する注意事項 改訂 #20110804

インテル® コンパイラーは、互換マイクロプロセッサー向けには、インテル製マイクロプロセッサー向けと同等レベルの最適化が行われない可能性があります。これには、インテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2)、インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3)、ストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。インテルでは、インテル製ではないマイクロプロセッサーに対して、最適化の提供、機能、効果を保証していません。本製品のマイクロプロセッサー固有の最適化は、インテル製マイクロプロセッサーでの使用を目的としています。インテル® マイクロアーキテクチャーに非固有の特定の最適化は、インテル製マイクロプロセッサー向けに予約されています。この注意事項の適用対象である特定の命令セットに関する詳細は、該当する製品のユーザー・リファレンス・ガイドを参照してください。

© 2013 Intel Corporation.無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Cilk、VTune は、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

JPN/1311/PDF/XL/SSG/KS Improve-Fortran-Code-Quality-with-Static-Analysis_Eval_Guide /Rev-083012