ET2018 設計検証ツールトラック

C/C++コンパイラをテストして妥当性を検証する

～ ユーザーまでもがコンパイラをテストすべき理由とは？ ～

FUJI SETSUBI

www.fuji-setsu.co.jp

Solid Sands

本日の内容

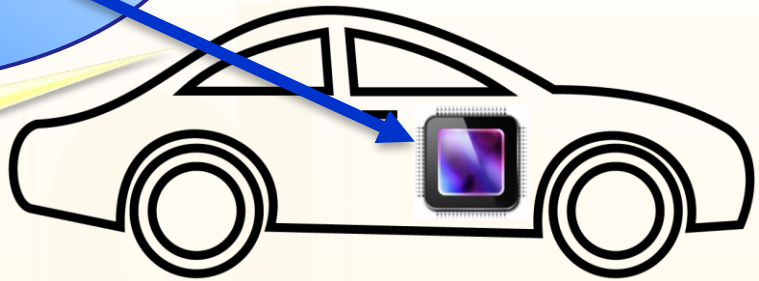
- C, C++コンパイラが正しいことを示すことがなぜ重要であるのか
- コンパイラのテストはどのようにして実施するか
- SuperTestの使い方
- 最新のトピック

コンパイラテストの重要性

ブレーキに
対する
Cソースコード

コンパイラは巨大で非常に複雑なソフトウエア
- 品質の検証には膨大な条件の組合わせによるテストが必要
- コンパイラメーカは全オプションの組合せをテストできない

C/C++コンパイラ
500万行規模



間違ったコードが生成されることによる被害が甚大

SuperTest™

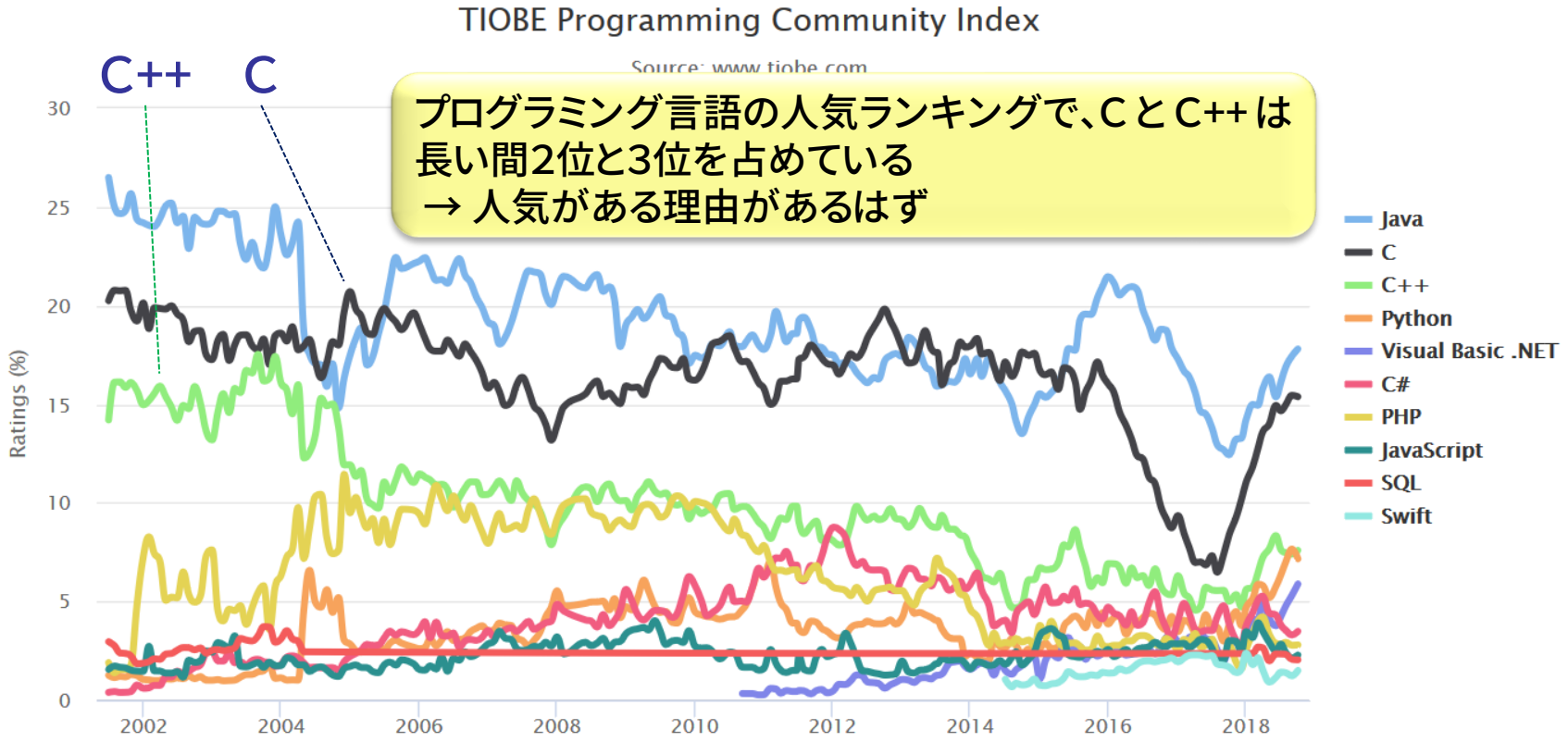
- 言語標準規格に対するテストをすることでコンパイラが正しいことを証明
- C, C++言語標準規格への適合性、正確性、堅牢性をチェックする300万件以上のテスト
- 30年以上の経験の積み重ねで開発
- 主にポジティブとネガティブテスト

・・・

コード生成機能を評価するテスト

診断機能を評価するテスト

C, C++コンパイラの重要性

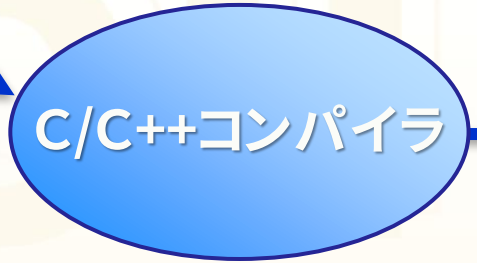


C, C++の特長

- 多くの異なるアーキテクチャに移植可能
- 基本システムなしのコンピューターでハードウェアにアクセスして実行可能
- 実行時システム不要（オーバーヘッドが可視、リアルタイム性）
- 高いパフォーマンス
- ISOによる標準化（多くのツールや見識あるエンジニアの存在）

コンパイラの仕事

```
int factorial (int n) {
    int res = 1;
    while (n > 1) {
        res *= n--;
    }
    return res;
}
```



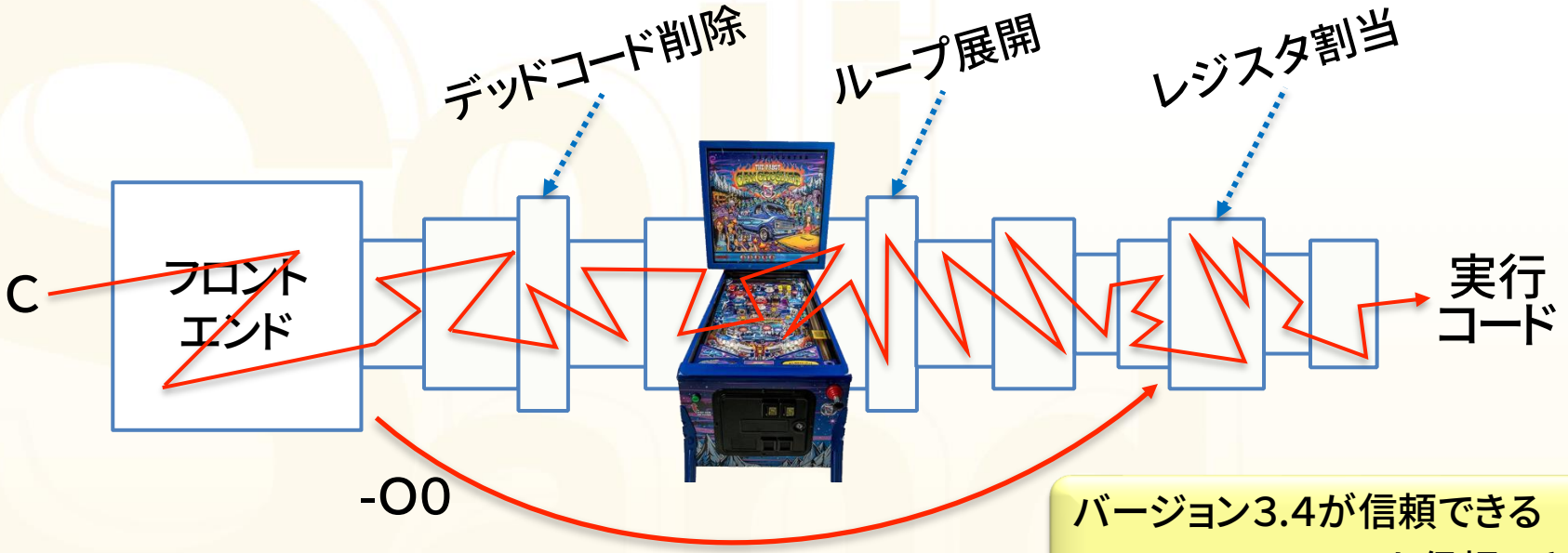

```
factorial:
    cmpl  $1, %edi
    jle   .L4
    movl  $1, %eax
.L3:
    leal  -1(%rdi), %edx
    imull %edi, %eax
    movl  %edx, %edi
    cmpl  $1, %edx
    jne   .L3
    rep ret
.L4:
    movl  $1, %eax
    ret
```

コンパイラの難しさ

- C, C++コンパイラのソースコードサイズ：500万行
- 開発期間：40年近く（GCC）、何千人ものエンジニア
- 複数のプロジェクトガイドライン
  - 安全には配慮されていない
  - カバレッジテストなし、MISRA準拠なし、・・・
- コンパイラ開発はいつも進行中
  - エラー修正、改良、機能追加 が常にある

コンパイラの内部構造

最適化では、プログラムはコンパイラの非常に多くの面に触れる
1つのステージでの小さな変更が、以降の流れを全く違ったものにすることもあり得る

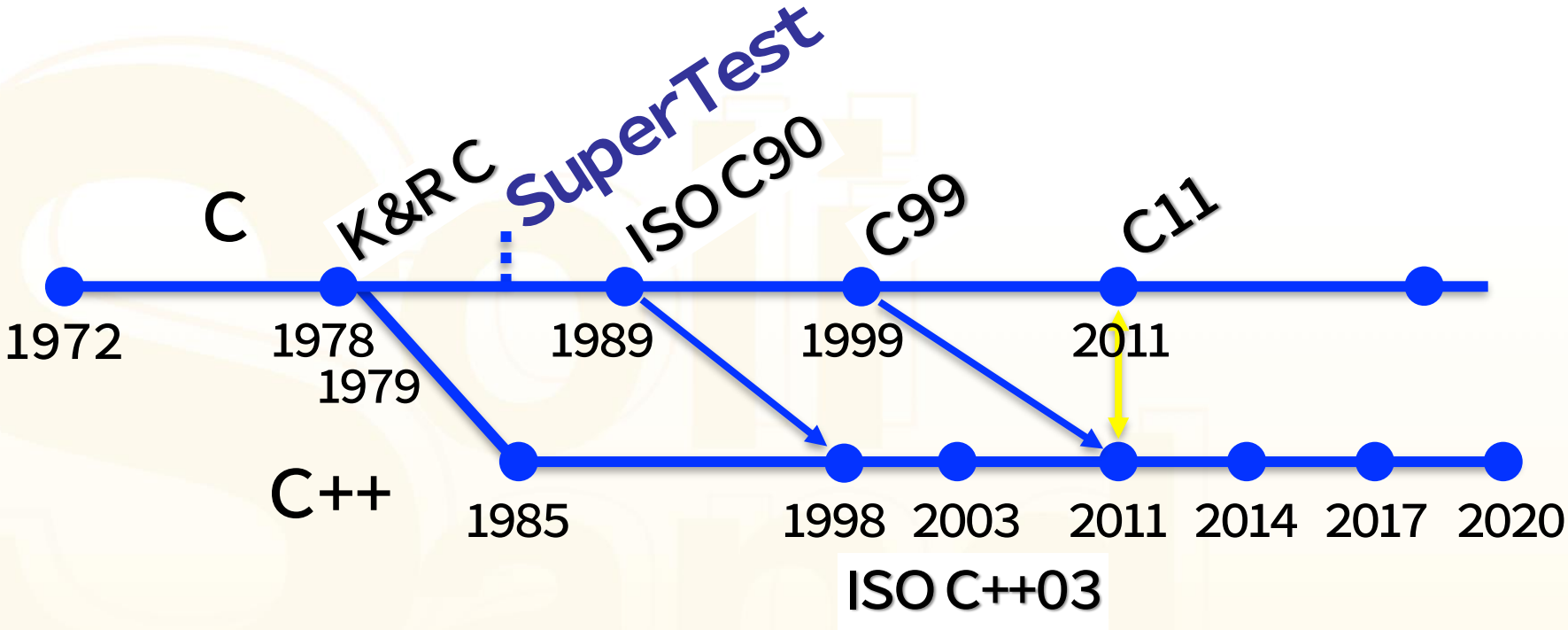


バージョン3.4が信頼できる
≠ 3.5も信頼できる

コンパイラの仕様 = 言語規格

- C, C++の歴史
- ISO標準:
  - C90, C99, C11, C++03/11/14/17
- 処理系定義の動作

CとC++の歴史



コンパイラがテスト可能である理由

- 明確に定義され、安定した仕様をもつ
- 単一入力、単一出力
- 決定論的
- コマンドラインでスクリプトの利用が可能

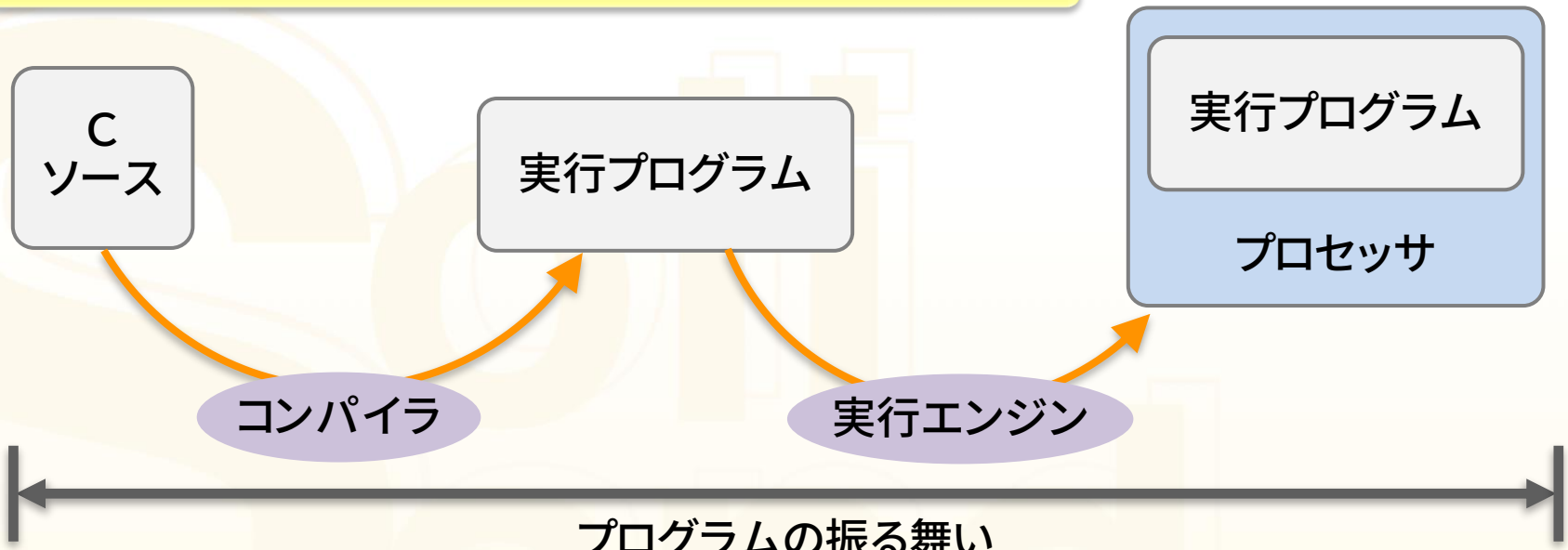
これらの性質によって、適切なテストスイートを書き、何年にもわたってそれを保守することが可能になる

本日の内容

- C, C++コンパイラが正しいことを示すことがなぜ重要であるのか
- **コンパイラのテストはどのようにして実施するか**
  - **コンパイラの振る舞いのテスト**
  - **正しくないプログラムに対する診断の検証**

言語仕様がプログラムの動作を定義

言語仕様はプログラムが実行されたときの動作を定義するだけ



プログラム実行の振る舞いをテスト
→ 各段階での振る舞いが正しいことを検証するテストを作成

ISO C99での例

- コンマ演算子
  - 構文規則

    式 ：
    代入式
    式 ， 代入式

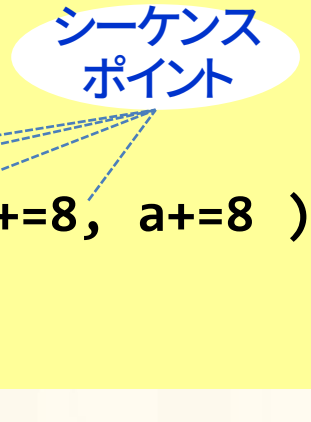
  - 意味規則　コンマ演算子は，左オペランドをボイド式として評価する。その評価の直後を副作用完了点とする。次に右オペランドを評価する。コンマ演算子の結果は，右オペランドの型及び値をもつ（94）。

テスト項目：評価順序、シーケンスポイント、・・・

SuperTestでのコンマ演算子のテスト

テストプログラム t2.c

```
void ge( int *p ) {
    *p = 2;
}

int test_it( void ) {
    int a, *p, r;
    p = &a;
    r = ( ge(p), a++, a+=3, a+=8, a+=8 );
    return r == 22;
}
```



不適合なプログラムと診断を出すもの

- **論理AND演算子**
  - **構文規則**

    論理ＡＮＤ式：
    ＯＲ式
    論理ＡＮＤ式　&&　ＯＲ式

  - **制約**　各オペランドの型は，スカラ型でなければならない。
  - **意味規則**　&&演算子の結果の型は，両オペランドの値が０と比較してともに等しくない場合は１，それ以外の場合は０とする。結果の型は，intとする。

    ビット単位の２項＆演算子と異なり，&&演算子は左から右への評価を保証する。

SuperTestでの演算子の優先順位のテスト

テストプログラム xspr2089.c

```
void test_it( void ) {
    int i = 1; j = 5;
    while (--i && j >>= 1) ;
}
```

先に評価される

コマンドラインからのコンパイル実行

```
$ cc -c xspr2089.c
xspr2089.c:33:29: error: expression is not assignable
        while (--i && j >>= 1)
                      ~~~~~~~~ ^
1 error generated.
```

正しいコンパイラでは構文エラーの診断を出す

ISO 26262によるコンパイラテストの形式化

- ISO 26262 8–11: ソフトウエアツール使用への信頼
- 特定のユースケースに対するコンパイラ認定のプロセスの記述

  ユースケースでコンパイラがどのように使用されるかを記述
  異なるコンパイラフラグでテスト

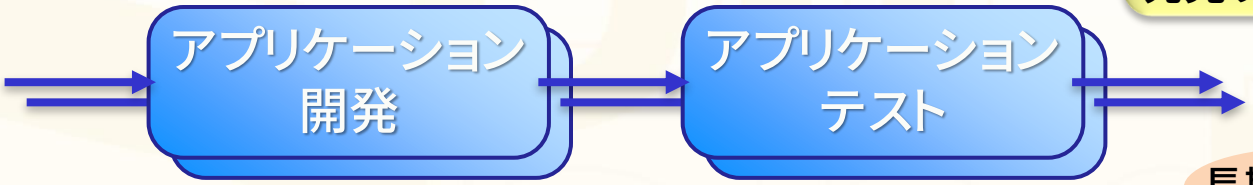
- 自動車産業界での実践

コンパイラ認定の利点：市場投入時間の削減

コンパイラ認定なしの場合：

アプリケーションテストは
すべての適切なエラーを
発見するには不十分

コンパイラ認定ありの場合：



コンパイラがテストをパスすれば、
すべてのケースでアプリケーション
によらずOK

長期間使用

異なるターゲット

異なるコンパイラ
バージョン

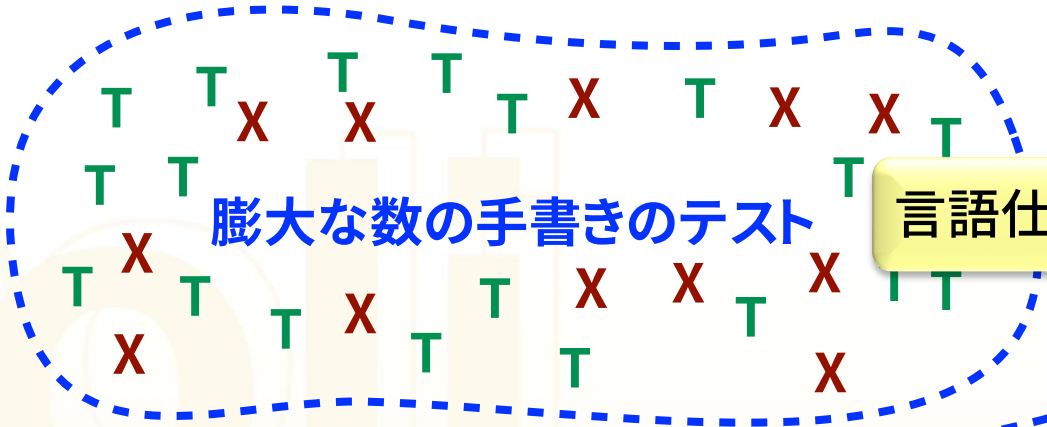
社内で継続的なコンパイラ認定を行うことの利点

- コンパイラ（バージョン）選定での自由度
  - コンパイラ改訂に対して最新を維持
  - コンパイラベンダーに縛られない
- ユースケース選定の自由度
  - コンパイルオプションやコンパイル環境適用の自由度

SuperTestによるテストの基本

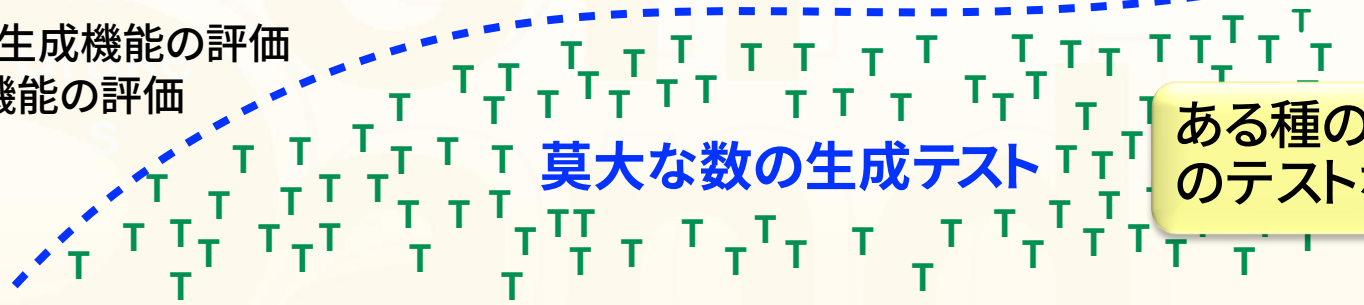
テスト
ドライバ

テスト
レポータ



言語仕様をカバー

Tはコード生成機能の評価
Xは診断機能の評価



ある種の算術演算
のテストなど

本日の内容

- C, C++コンパイラが正しいことを示すことがなぜ重要であるのか
- コンパイラのテストはどのようにして実施するか
- **SuperTestの使い方**
- 最新のトピック

SuperTestの動作条件

ホストコンピュータ

（クロス）コンパイラ
1 GB ディスク

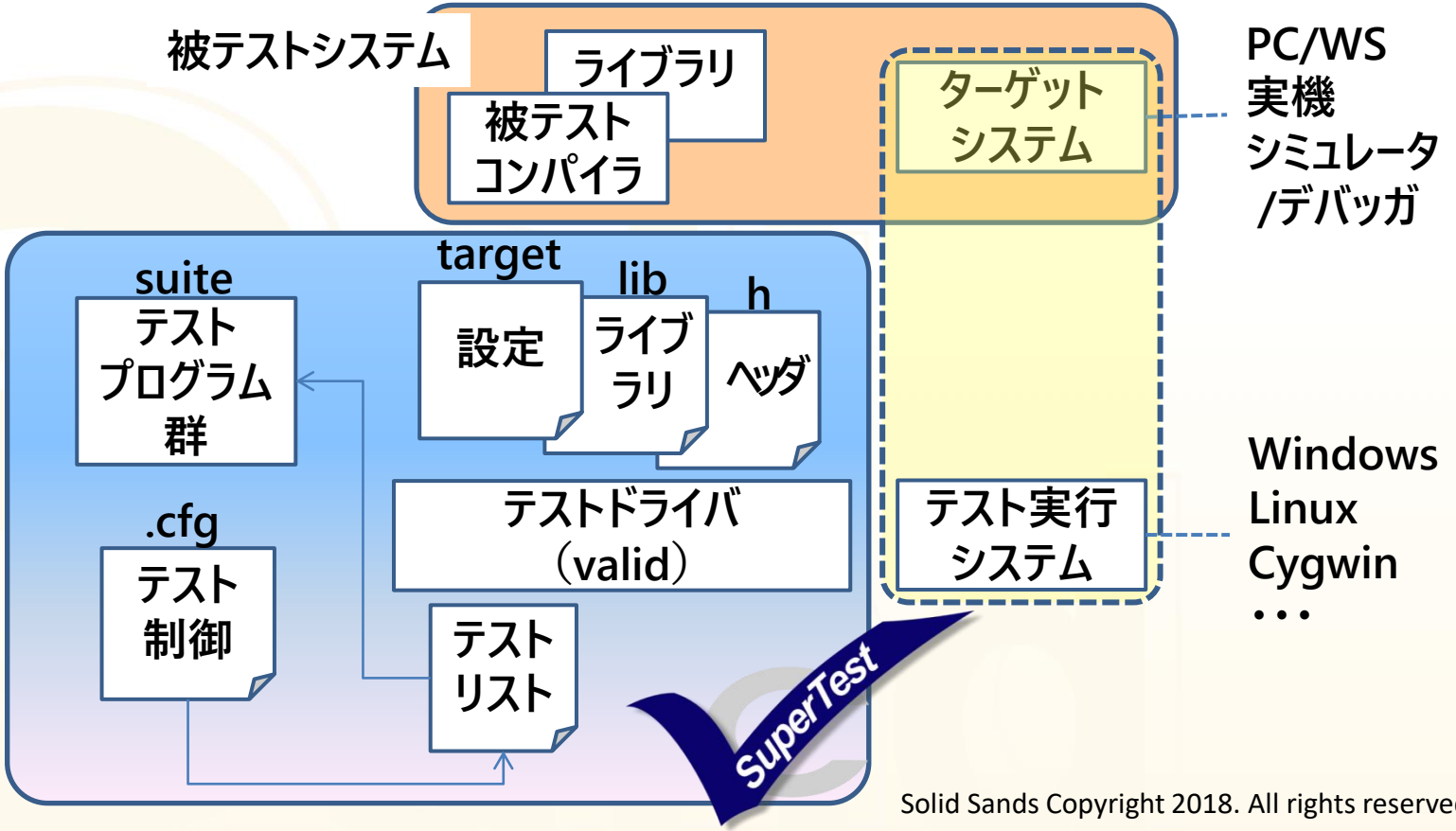
- POSIX 環境
  - Linux
  - Mac
  - Solaris / HP-UX /・・・
  - Windows + Cygwin
- ネイティブ Windows プラットフォーム

実行環境

- ネイティブ
- ターゲットボード
- シミュレータ

4KB RAM
ホストへの終了値

SuperTestの概要



実行システム上でのテスト

- Cygwin上の gcc をテスト
  - コンパイラの指定 … cc
  - オプション
  - ワークエリア

```
CC=cc
CFLAGS=''
LIBFLAGS='-lm'

TARGET=target/

TESTLIST=demolist

LOG=LOG
```

- テストリストで指定されるテストプログラムをバッチ実行
- ログ出力

ログ出力の例

```
TESTING: suite¥2¥2¥1¥1¥t100.c

   Trigraph sequen
+++ stderr +++++++
t100.c:10: parse e
t100.c:12:16: warn
t100.c:24:33: warn
t100.c:24:33: miss
t100.c:12:16: poss

   Compilation fai

RESULT: 2¥2¥1¥1¥t1
```

```
TESTING: suite¥3¥3¥2¥3¥x5.c

   const qualified
+++ stderr +++++++
x5.c: In function
x5.c:27: warning:
         read-only

   Compilation inc
                with

RESULT: 3¥3¥2¥3¥x5
```

```
TESTING: suite¥C99¥6¥7¥8¥t7.c
RUNFAIL
+++ stdout ++++++++++++++++++++++
    Partial override

    line 54: failed at:
    y.t.k == 42

    line 56: failed at:
    y.t.a[0] == 18

RESULT: C99¥6¥7¥8¥t7.c     FAILED
```

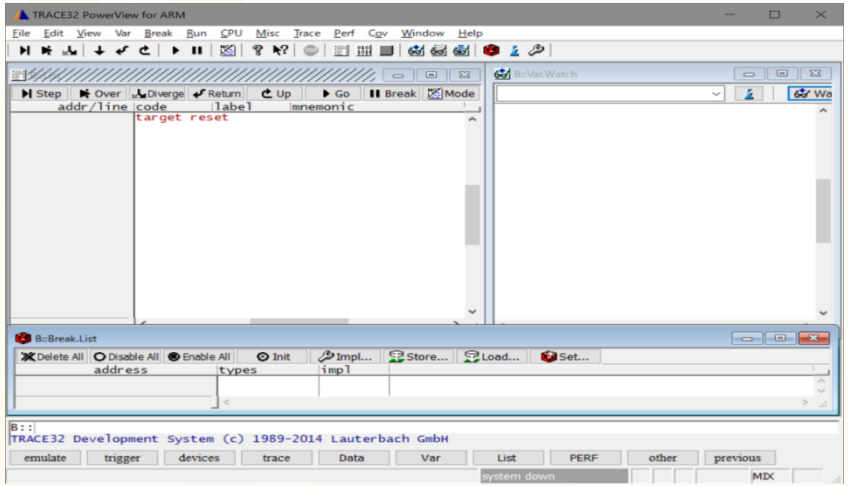
ターゲットシステム上でのテスト

- Windows上のarmクロスコンパイラ
  - コンパイラの指定 … コンパイラパス
  - オプション
  - ワークエリア

```
CC='/c/SysGCC/arm-eabi/bin/arm-eabi-gcc.exe'
CFLAGS='-std=c99 -O0 -g3 -mcpu=cortex-a9'
LIBFLAGS='-L~/SuperTestDemo/SuperTest/LOGsim/work/lib -lm'

TARGET=target_sim

TESTLIST=testdemolist

LOG=LOGsim
```

動作結果

- 実行: armシミュレータ TRACE32
  - ローターバッハ社 デバッガ/シミュレータ
  - シミュレータ用スクリプト
  - テストプログラム出力の代替機構

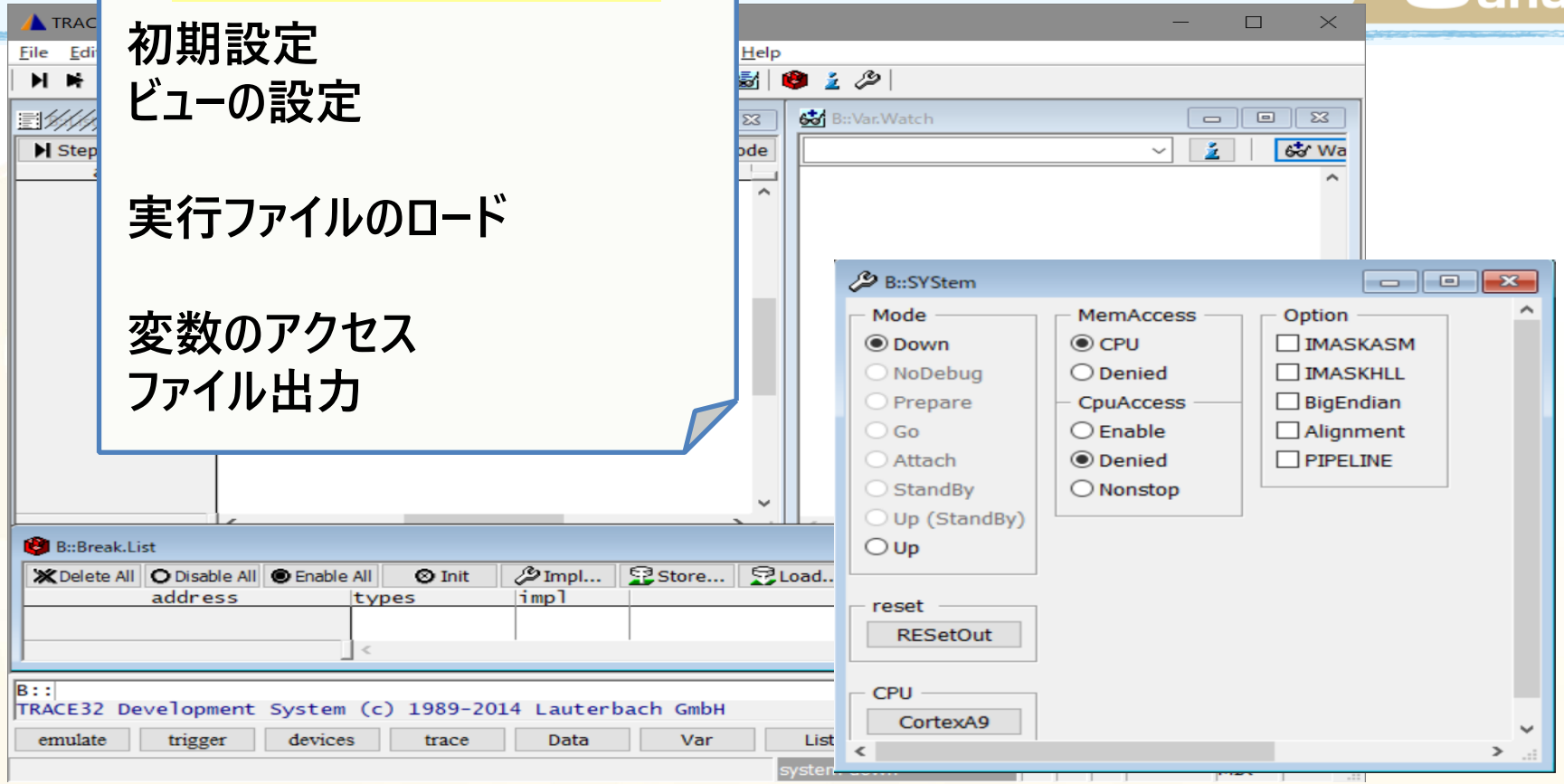


TRACE32バッチファイル

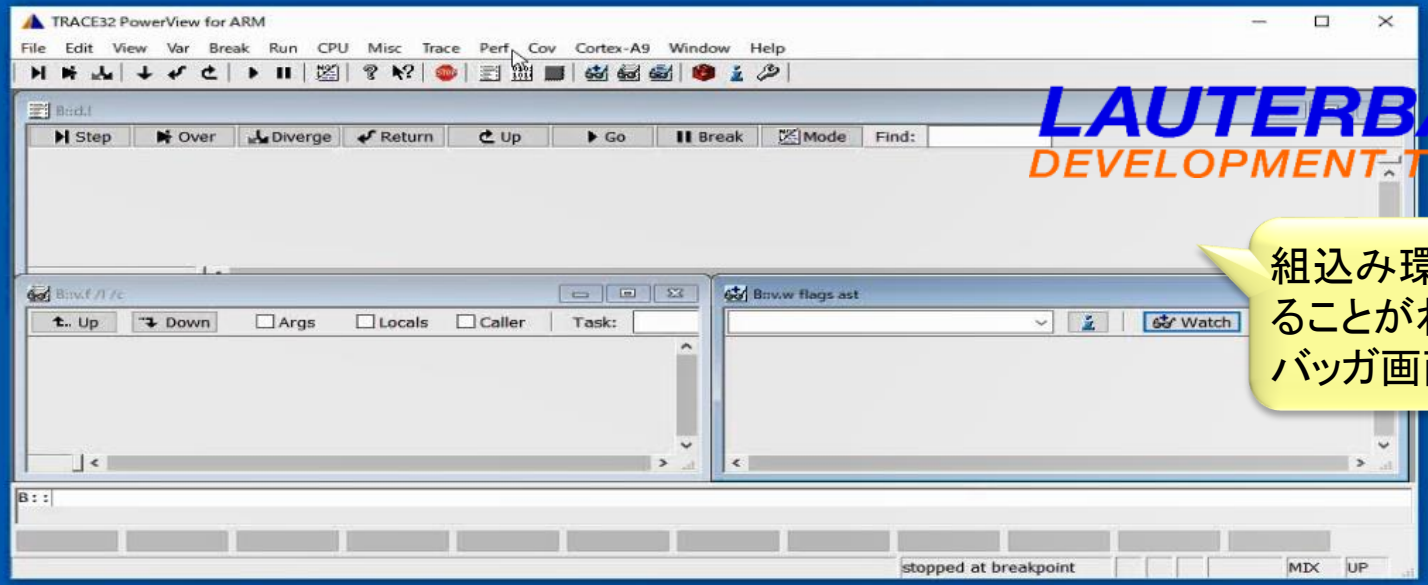
初期設定
ビューの設定

実行ファイルのロード

変数のアクセス
ファイル出力



動作結果 … シミュレータでのコマンド実行



LAUTERBACH DEVELOPMENT TOOLS

組込み環境でテスト実行していることがわかるように、わざとデバッガ画面を起動させています

デモ:ローータバッハ社デバッガで実行

デモ動画公開：https://www.fuji-setsu.co.jp/demo/SuperTestDemoLauterbach.wmv

本日の内容

- C, C++コンパイラが正しいことを示すことがなぜ重要であるのか
- コンパイラのテストはどのようにして実施するか
- SuperTestの使い方
- **最新のトピック**

SuperTestのバージョンアップ

- Windowsネイティブのサポート
  - Pythonによるテストドライバの実装
- テスト結果から言語仕様へのトレーサビリティの強化

SuperTest先進のトピック

- 最適化のテスト
- ABIテスター
- 算術演算 Depth スイート
- MISRA C
- 機能安全でのコンパイラ認定

コンパイラによる最適化

- プログラムの実行速度の改善
- 振る舞いは変えない
- 言語規格では・・・
  - この規格における意味規則の規定では，最適化の問題を無視して抽象計算機の動作として記述する。
- Cの非機能の振る舞いを改善。コンパイラの重要な機能
- どのようにして最適化をテストするか？

- **最適化なしの場合**
  - ソースコードレベルの単体テストで100%のMC/DCカバレッジ
  - コンパイルされたオブジェクトコードレベルでも100%のカバレッジを確認できる
- **最適化があると・・・**
  - 場合分けされ、生成コードが増大
    → カバレッジのため追加テストが必要
    - 機械語レベルでの解析なしに、テストを作ることは不可能
    - 場合分けはソース・コンパイラ・オプション・CPUに固有
    - 冗長なコードの可能性

SuperTestの最適化対応

- SuperTestに対して最適化のテストを行う技法を開発中
  - 機械語でのMC/DCカバレッジに関する作業
  - コンパイラ自体の構造カバッレジに関する作業
- 現状のスイートでも最適化のテストに有効

呼出し規約に対するABIテスター

- 呼出し規約: 関数間での引数の渡し方などの定義

```
typedef struct{double v0;}t0;
typedef unsigned char t1;
extern void func(t0 v0, t1 v1);

int main (void){
  t0 v0;
  t1 v1;
  v0.v0 = 9071595.0;
  v1 = 234U;
  func(v0, v1);
  return 0;
}
```

Caller

```
#include <assert.h>
typedef struct{double v0;}t0;
typedef unsigned char t1;

void func(t0 v0, t1 v1)
{
  assert (v0.v0 == 9071595.0);
  assert (v1 == 234U);
}
```

Callee

別々のコンパイラ、同じコンパイラの別バージョンでコンパイルしてテスト可

実装依存の算術演算

- Cのデータモデルは実装で定義：intは、16/24/32/・・・ビット
- データモデルが定められると、言語の定義により算術演算の動作に対して厳密にルールが適用：昇格、変換、オーバーフロー、符号
- SuperTestでは特定のデータモデルごとに異なるテストスイートを生成し、Depthスイートとして用意

機能安全に対するコンパイラ認定

- 目標：セーフティクリティカルなアプリケーションへのコンパイラの使用に対する信頼を得る
  - 特定のユースケース（アプリケーション）
    例えば、最適化レベル -O1 で得られた認定は -O2 では有効ではない
  - 言語仕様に対する検証によるもの
  - コンパイラの欠陥に対して緩和策を定義する
    そのコンパイラを使用するアプリケーション開発者が考慮すべきもの

Solid Sans社によるコンパイラ認定サービス

- ISO 26262に従った「ツール認定」
- 検証手順と「ユースケース」を協力して定義
  - ユーザーが用意するもの
    CPU、コンパイラ、デバッガ、コマンドラインスクリプト、
    コンパイラのオプション設定を含む事例
- テスト実行、結果の分析、緩和策の定義
- 生データを含んだ完全なテストレポート

SuperTestの成果

- 同じコンパイラを1000本以上使用
  - バージョンアップごとにチェックし、使ってはいけない最適化などの社内ルールを施行。過去バージョンに戻すような問題が起こっていない
- ABIテスター機能でABI規則への準拠を確認
  - 他のサプライヤーの製品と繋がることによる問題の切り分け
- ライブラリのテスト
  - C99にC89のライブラリが混在して数学関数で問題になるところを事前に回避
- Depthスイート機能
  - アーキテクチャ固有のデータ長（例えばcharもintも24ビット）に合わせて演算精度のテストができる

まとめ

- SuperTestは日本を含む世界中で使用
- SuperTestはコンパイラ開発者だけでなく、コンパイラユーザーにとっても容易に使用
- Solid Sands社は、ISO 26262のような機能安全の領域で専門技術とサービスを提供

SuperTest is the best test-suite for C and C++ compilers and libraries

コンパイラユーザーもSuperTestを採用

ANDES TECHNOLOGY
DENSO
ARM
SONY
synopsys®
Ambarella™
Qualcomm
Peloton
RENESAS
HIGHTEC
nexteer AUTOMOTIVE
a leader in intuitive motion control
cadence
CIRRUS LOGIC®
CEVA®
TEXAS INSTRUMENTS
BOMBARDIER
ERICSSON
NVIDIA
WIND™
AN INTEL COMPANY
AbsInt
NXP
ROHM SEMICONDUCTOR
ANALOG DEVICES
Embedded Office
SAMSUNG
codasip
ZÖLLNER
SIGNAL SYSTEM TECHNOLOGIES
SILEXICA
Melexis
INSPIRED ENGINEERING
octasic
diffblue
AI for Code
Green Hills SOFTWARE
DRÄXLMAIER
SOMNIUM
XMOS®
HUAWEI

車載システムの機能安全のリスクを軽減

実はコンパイラを検証する必要があるのは、コンパイラ開発者だけではない。コンパイラによってアプリケーションコードに不測のエラーが混入されることがないように、ソフトウエア開発者であってもコンパイラの品質を意識する必要がある。特にセーフティクリティカルな製品では、その傾向が顕著になる。

株式会社デンソーも、この問題を解決するコンパイラテストと検証用のパッケージとして、Solid Sands 社のSuperTest を支持している。

コンパイラで実績済みのソースコードを再利用することは、ソフトウエアや製品の品質を維持する最善策のひとつである。しかしながら新たに製品系列を展開する場合に、新しく生成されるソースコードが、コンパイラの欠陥を表面化させる可能性がある。さらにソフトウエア開発担当者ごとで異なる様々なコードスタイルによって、コンパイラの欠陥が浮き彫りにされることもある。また一方、コンパイラに潜在する問題が露呈するのは、新規のソースコードをコンパイルする場合に限らない。

顧客事例　https://www.fuji-setsu.co.jp/products/SuperTest/#customers

各種資料を公開しています

・コンパイラユーザからよくある質問とその答え

https://www.fuji-setsu.co.jp/files/SuperTestFAQs.pdf

・ポジティブテストによるコンパイラバックエンド（ジェネレータ）のテストについて

https://www.fuji-setsu.co.jp/files/JP_Code_generator_validation.pdf

・C、C++コンパイラの tool qualification の効用

https://www.fuji-setsu.co.jp/files/QualificationBenefitsSuperTestf_JP.pdf

・コンパイラのテスト結果からアプリケーションコードの品質を改善

https://www.fuji-setsu.co.jp/files/CompilerTestResults.pdf

Thank You!

FUJI SETSUBI

Solid Sands