コンパイラバックエンドのテスト

ユーザーまでもがコンパイラをテストすべき理由とは？

Code Generator Development and Validation

Why SuperTest should be your first choice for compiler code generator development and validation!

By Marcel Beemster

コードジェネレーターの開発と検証に SuperTest

コンパイラを構成する要素のうち、バックエンドとも呼ばれるコードジェネレーターには、専用に設計された特別なテストが必要になります。これは SuperTest™にとっては得意中の得意ですが、主にフロントエンドが担う言語規格への準拠を確認するためだけに設計された他社製品には不可能なことです。またコンパイラ開発プロジェクトのごく初期段階から使い始めることができるのも、他社製品にはない特長です。さてその理由とは……

SuperTest にはコンパイラを調べる三つのテクニックが組み込まれています。専門家のノウハウが詰め込まれたテストコード、Depth Suite、ABI-Tester です。

SuperTest はライブラリを簡単にテスト対象から外すことができます。開発途中かも知れないライブラリには依存せず、レポート生成に最小限のインターフェイスしか必要としません。このため SuperTest はコードジェネレーター開発プロセスのごく初期の段階から使い始めることができるのです。

新規開発されるコンパイラの大部分が GCC あるいは CLANG+LLVM コンパイラのいずれかに依存する今日では、コードジェネレーターのテストは極めて重要な領域です。フロントエンドに関しては、各々のコンパイラ間で差異はあったとしてもごくわずかなものです。このため多くのコンパイラでほぼ同じフロントエンドが共有されることになり、結果的に莫大な数の開発者や SuperTest ユーザーによって充分なテストを受けることになります。

それに対してコードジェネレーターはターゲットごとに異なるものが開発され、使用されます。また多くのケースで特注品あるいは特定アプリケーション専用の CPU を対象にします。そういったコンパイラは開発者もユーザーも少数ですから、欠陥の発見と修正も貧弱なエコシステムに頼らざるを得ません。これが意味するところは、コンパイラ品質を維持するには、高品質のツールによるコードジェネレーターの検証が不可欠ということです。SuperTest こそがそのツールです。

中間レベルの最適化とコードジェネレーター

コードジェネレーターはターゲット CPU ごとに必ず書き直される部分です。コードジェネレーターには重要な三つの役割があります。命令の選択、レジスタの割当て、スケジューリングです。その他ターゲット固有の機能、たとえば CPU がサポートしない中間表現の演算を他の命令の組み合わせでエミュレートしたり、制御フローをプレディケーションに変換したりもします。

コードジェネレーターがバックエンドとも呼ばれる理由は、フロントエンドと中間レベル（たいていはターゲットに依存しない）の最適化の後ろ、すなわちコンパイラの最終段階に位置するからです。テストが意図した通りに最終段階のコードジェネレーターまで到達するには、中間レベルの最適化を無効化しないといけません。さもないと、中間レベルの最適化によってテストされるプログラム構造が意図せず別の形に変換されてしまい、想定したターゲット命令のテストに失敗してしまうかも知れません。オプションを指定しない場合でさえ、コンパイラによっては最適化を実行してしまうことに注意してください。もし可能なら切っておくのが良いでしょう。

またテストの後半では、念のため最適化を有効化して実行することも必要です。中間言語の最適化は、コードジェネレーターに向けてひねくれた球をはね返すピンボールマシンのようなものと考えてください。テスト材料としてはある意味とても優秀ですが、球がどこに当たるのか、つまりコンパイラのどの部分をテストするのか、正確にコントロールすることは容易ではありません。

専門家のノウハウが詰まったコンパイラテスト集

コードジェネレーターのエラーに対する第一の防衛線は、この分野に精通した経験豊富な専門家のノウハウを詰め込んだ膨大な数のテスト集です。SuperTest は CoSy® を含む、何世代にもわたるコンパイラ開発システムとの密接な連携を通じて開発されました。CoSy は 100 を超える異なるターゲットアーキテクチャーに使われてきた、高度なターゲット切り替えが可能なコンパイラ開発システムです。コードジェネレーターの開発と検証は常に CoSy の最重要課題です。
このため SuperTest にはコードジェネレーター専用のあらゆるテストが含まれています。これらの開発は、コードジェネレーターに関係するすべてのプログラミング構造や演算子や型の洗い出しから始まりました。そしてカバレッジ分析を用いて最適化のオンとオフ両方で漏れがないかを確認しました。さらに、コードジェネレーターやコンパイラのその他の部分の開発中に見つかったエラーも SuperTest に取り込まれました。

ターゲット固有の算術処理のための Depth-Suite

第二の防衛線は、ターゲットの算術処理固有の DepthSuites です。そのほうが都合良いといういくつかの理由があって、C 言語は算術型のビット数を厳密に規定していません。たとえば「int 型は 16 ビット以上」というような決まりがあるだけです。だからと言って C 言語の算術処理の定義が緩いという訳ではありません。それどころか実装時には算術型のサイズと精度が明確に定義されていなければなりません。それらは実装定義データモデルの一部です。

つまり、C 言語の実装において算術処理のビット数は自由に決めて良いということですが、いったん決めた後はそれを明確にして厳守する必要があります。そしてその選択に基づき、算術処理の挙動が決まります。

例えば、二つの符号なし整数の加算がオーバーフローしたとき、結果はその符号なし整数の最大値より 1 大きい数による剰余によってラップアラウンドします。ここで、「符号なし整数の最大値」が実装定義のパラメーターです。

C 言語の算術演算の定義は非常に正確ですが、その実装が定義する属性への依存のために、汎用的なテストスイートでは正確な確認が困難になります。

ところが SuperTest ならこれが可能です。SuperTest には 30 以上の DepthSuites が含まれています。これには、整数と浮動小数点の両方の膨大な数のデータモデル固有の算術処理テストなどがあります。これらのテストはすべての算術型と演算子と値の組み合わせ、特に汎用的なテストでは対処できないコーナーケースも網羅しています。

DepthSuites に含まれないデータモデルについてはどうでしょうか？ その時はご要望に応じて新たに作成いたします。

DepthSuites のテストは最適化あり、なしの両方で実行できます。最適化なしだとコードジェネレーターの算術処理の実装にダイレクトにマップされます。先に述べましたように、これは望ましいことです。最適化ありだと、中間レベルではありますがターゲット固有の定数畳み込みに対する良いテストになります。なぜなら定数畳み込みもまたターゲットの算術処理ルールを厳守しなければならないからです。

呼出し規約とレジスタ割当てのための ABI-Tester

第三の防衛線は、ABI（Application Binary Interface）-Tester です。ABI-Tester はターゲット固有の呼出し規約の実装の検査になるように設計されています。ABI-Tester はそれぞれ別のファイルに関数のペアを生成します。呼び出す側のコーラーと呼び出される側のコーリーです。ABI-Tester はコーラーからコーリーに渡される、設定可能であるランダムなパラメーター系列を生成します。両関数はセルフチェックするようになっていて、コーリーは正しい値が渡されたか確認し、コーラーはコーリーの戻り値を確認します。

呼出し規約の実装はレジスタの割当てと最適化に密接に関連しています。レジスタの割当てもまたターゲット依存であり、最適なレジスタの割当てはコードのパフォーマンスに関わるので、これは複雑な問題です。これは本来の使用目的ではありませんが、レジスタの割当てのテストにも ABI-Tester が役立ちます。

最適化を有効にしても呼出し規約自体には影響しません。しかし関数呼び出しの×レジスタ圧力×には影響を与えます。最適化により、より多くのレジスタが使われることになり、割当てアルゴリズムはどの値をどのレジスタに格納するか、判断が難しくなります。そこでここでもまた、最適化なしでコンパイラの呼出し規約とレジスタ割当ての開発を始めることは意味があります。その後最適化を有効にしてハードルを上げればよいのです。

ABI-Tester の効用は、呼出し規約の内部実装のテストだけに限定されません。関数ペアを二つの別のコンパイラにかけて、両者の呼出し規約が一致しているか確認することもできます。

あるいは関数ペアをそれぞれ同じコンパイラの別のバージョンにかけて、呼び出し棄却の実装に変更がないことも確認できます。

コードジェネレーターの欠陥からあなたを守る SuperTest

コンパイラには必ずコードジェネレーターがありますから、SuperTest に含まれる実行形式のテストはすべて、結局のところコードジェネレーターをテストしていることになります。つまりコードジェネレーターのカバレッジに関してはこれで充分です。

それに加えて SuperTest にはコード生成エラーに対する三重の防衛線があります。第一の防衛線である達人テスト集は、コードジェネレーターに求められる個々の機能に対するチェック機能を持っています。これらのテストは複数のターゲットを切り替え可能なコンパイラシステムの開発のために作成され、そのプロジェクトを通して改良されました。

次に第二の防衛線である DepthSuites は、コードジェネレーターのすべての算術処理と型とそれらの組み合わせをテストできます。DepthSuites はターゲット依存の算術処理のレアケースを確認します。コンパイラの固有の実装依存のデータモデルに対して生成されるからです。

そして第三の防衛線である ABI-Tester は、呼出し規約の実装を確認するテストを生成します。さらに呼出し規約の実装と密接な関係があるターゲット依存のレジスタ割当てに対するストレステストとしても機能します。

最後に、SuperTest はコードジェネレーター開発の初期段階から使っていただけるように作られています。

これが、コンパイラのコードジェネレーターの開発と検証には SuperTest を選ぶべき理由です。

FUJI SETSUBI

富士設備工業株式会社 電子機器事業部

www.fuji-setsu.co.jp