社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ＳＥ－156 (2)
2007－EMB－ 5 (2)
2007／5／28

アセンブラのモデル検査におけるモデルの自動生成について

宇佐美雅紀, 藤倉俊幸

イーソル株式会社リサーチ＆コンサルテーションサービス部

アブストラクト

アセンブラプログラムのモデル検査についての知見と, アセンブラプログラムの検査モデルの自動生成について報告する. アセンブラプログラムのモデル検査では, CTL 式を使用した到達可能性検査が有効である. また, 検査モデルを正確に生成するために, モデル変換スクリプトによるモデルの半自動生成が有効である.

Automatic model generation of the assembler program for model checking

Masanori USAMI, Toshiyuki FUJIKURA

Research & Consultation Services Group, eSOL Co., Ltd.

Abstract

This paper describes about automatic model generation of the assembler program for model checking and the finding of the model checking of the assembler program. In the model checking of the assembler program, the reachability check that used CTL (Computation Tree Logic) is effective. Moreover, the model's semiautomatic generation with the model conversion script is effective to generate the model accurately.

1. はじめに

組込みソフトウェアの開発においては, RTOS の開発, 移植, デバイスドライバの開発等, アセンブラプログラムを必要とする場面が存在する.

本研究では, アセンブラプログラムの例として, RTOS のディスパッチャ部分を対象としてモデル検査を行った. C, JAVA では, これらの言語を対象とするモデル検査ツールが存在する[2][3]が, アセンブラプログラムを対象とするモデル検査ツールはない. 本研究では, モデル検査ツールには, CTL 式, LTL 式が使用できる NuSMV[1]を使用した. このモデル検査を通じて, アセンブラプログラムのモデル検査に関する知見を得たのでこれについて述べる. また, 検査モデルをアセンブラソースコードからモデル変換スクリプトを用いて半自動生成することで, 検査モデルとアセンブラプログラムとの変換についての知見を得たので, これについて述べる(図 1).

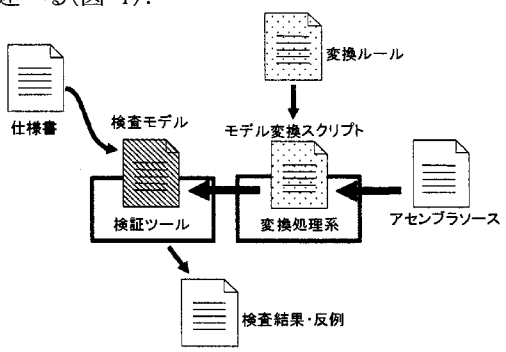

図 1アセンブラプログラムのモデル検査手順

2. モデル検証手順

アセンブラプログラムのモデル検査は次のように行った.

検査モデルは, アセンブラソースコードからモデル変換スクリプトを用いて, 半自動で生成した. これにより, モデルの正確性を確保し, また, 検査モデルとアセンブラプログラムとの変換に関する知見を得た. 検査モデルの生成については, 4章で述べる. 検査モデルは, レジスタの動作を忠実にトレースしたレジスタレベルのモデル(以下, レジスタレベルモデル)と, これを抽象化したモデル(以下, 論理レベルモデル)を作成した. 当初, レジスタレベルのモデルを作成して検査を試みたが状態爆発が起こり, 検査を実施することができなかった(表 1). そのため, モデル化するレジスタ数を減らすなどしたが, 状態爆発を回避することができなかった. 結果的に, アセンブラプログラムと外部環境とのインターフェースとなるグローバル変数, C 言語関数レベルでの抽象化を行った論理レベルモデルを作成した(図 2).

現状のアセンブラプログラムによるモデル検査では, このような抽象化は必須であると考えられる.

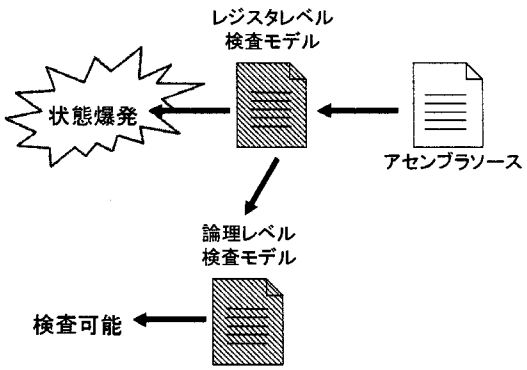

図 2モデル作成の流れ

表 1 検査に使用した計算機環境

CPU	32bits
動作周波数	3GHz
メモリー	1GB

3. アセンブラプログラムのモデル検査の特徴

通常アセンブラプログラムは, 組込みソフト全体の一部分であり, アセンブラプログラムだけを取り出してモデル検査を行なうには注意すべき点が 3 点あった.

アプリケーション

1 点目は, RTOS のディスパッチャ部分において, 二つのタスクから交互にタスクの起動とスリープを繰り返すような動きは組込みソフトでは普通に起こることである. ディスパッチャ部分のみを検査対象とした場合, この様な動きはモデル検査ツールによって無限ループと判定されてしまう. 同様のことは, 割り込みが連続して入るような場合にも起こる. このような動作が, 正しい動作か否かは, RTOS を利用するアプリケーションも含めて判断する必要がある. 一般のモデル検査では, アプリケーション部分もモデル化することで対応する. しかし今回は, 部分のモデル化は行なわずに, シナリオを動作制約としてモデル化することで対応した. 図 3 はシナリオの例である. この例では, タスク exit を∞回呼出さないようにするために, タスクが減少するシナリオを導入している.

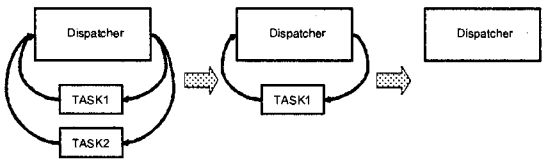

図 3シナリオの例

2点目は, 検査式としてCTLが有効に使えることである. 動作パス全体を検査対象とするLTL ではなくCTL を検査式として採用することで, 特定の動作パスの存在を確認することがで

きる. すなわち, 網羅的検査としては不十分であるが存在オペレータ E を使用できる CTL 式を使用することで, 正常系の動作パスの存在と特定のエラー状態への動作パスが存在しないことを示すことができる.

3点目は, アセンブラプログラムには内部にモジュール構造が存在しないことである. モジュール構造が存在すれば, モデルを階層化するなどして, 状態爆発を回避することが可能な場合もある. しかし, アセンブラプログラムではそれができずに, 状態爆発回避のためにモデル全体の抽象化を使用して論理レベルモデルを作成した. 論理レベルモデルでは, 状態爆発を回避することは出来たが, アセンブラソースコード検査ではなく, 設計レベルでの検査が中心となる.

4. モデルの自動生成

アセンブラプログラムの検査用モデルは, アセンブラソースコードからモデル変換スクリプトを利用して, 半自動で生成した. 自動生成により, 検査用モデルの正確性を確保することが可能になる.

モデル化の手順を図 4に示す.

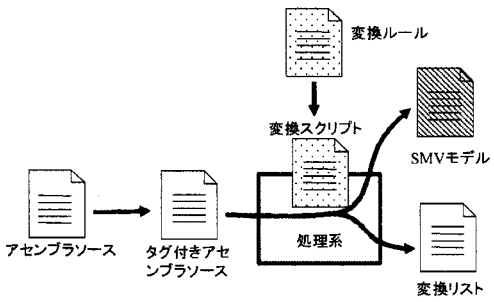

図 4モデル化の手順

アセンブラソースコードを, モデル変換規則を実装したモデル変換スクリプトにより, 検査モデルに変換する.

このとき, 処理すべきラベルの指定, C 言語関数の呼び出しであることの指定, 外部変数の指定, 構造体のメンバーの指定などの情報をソースコード上のタグとして埋め込むことにした(図 4中のタグ付アセンブラソース). これは手作業で行った. タグは, アセンブラ命令や, マクロ命令と容易に区別が出来るように, すべて##で始まるものとした. また, タグは必ず行の先頭におかなければならないものとした.

タグを埋め込んだアセンブラソースコードをモデル変換スクリプトで処理することで, 検査モデルを生成する. 検査モデルの生成と同時に, アセンブラソースコードの各行に asm_line 番号(後述)を併記した変換リストファイルを生成した. この変換リストは, 手作業によるモデルの変更, およびデバッグ時に役に立った.

モデル変換の基本的な考え方は, 図 5のようになる.

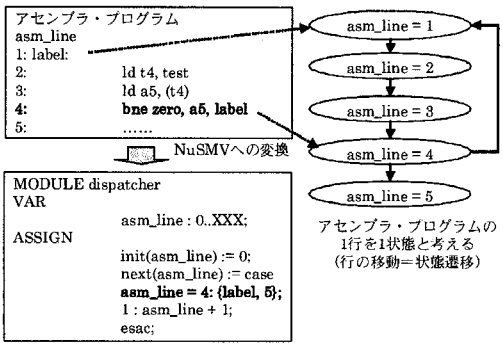

図 5 モデル変換の考え方

本研究のモデル変換では, アセンブラプログラムの 1 行ずつを1つの状態と考える. プログラムの実行, すなわち行の移動がモデルの状態遷移に対応する.

図 5のアセンブラプログラムには便宜のために行番号を記した. 各行番号が1つの状態を意味する. これを NuSMV モデルでは, 変数 asm_line で表す. アセンブラプログラムが 1 行実行されるごとに, asm_line は1つずつ増える. アセンブラプログラムのジャンプ命令, 分岐命令が現れると指定の状態へ状態遷移する(ここでは, asm_line = 4 から asm_line = 1 への遷移). また, 図のように分岐命令が条件分岐である場合は, asm_line + 1 の状態, および分岐

命令の指定先の両方への状態遷移が可能になるようにモデル化する(図中の NuSMV モデル太字部分).

さらに, レジスタやメモリの動作を忠実にモデル化することでレジスタレベルの検証モデルが生成できる.

また, アセンブラソースコード中に埋め込まれたタグを処理することで, C 言語関数やグローバル変数のモデル化を行う. 本稿では, タグの処理ルールはモデル変換スクリプトに内蔵したが, 処理ルールの設定により生成されるモデルは自在に制御が可能になる. これにより, 環境モデルも含めて, 自動変換でのモデル化が可能になる.

論理レベルモデルは, レジスタやメモリの動作をモデル化せず, 論理レベル専用に環境モデルの変換規則を実装することで実現した.

5. 検査項目

ソースコードに対応した到達可能性検査とデッドロック検査をおこなった. また, ソースコードとは別に仕様書から検査項目を抽出し, 検査式に変換して検査をおこなった.

当初, 検査環境としてタスク部分のモデルおよび割り込み例外処理の部分のモデルを作成し, これをディスパッチャのモデルと合成し検査環境とする予定であった. しかし, すでに述べたようにディスパッチャのみで状態爆発が発生してしまったので, 外部環境モデルと合成することは出来なかった. 別モジュールとして外部動作をモデル化すると合成時の状態数が相乗的に増えてしまうためである.

検査項目は, モデル検査一般でおこなわれる NuSMV ツールコマンドによるもの, およびソースコードから読み取ったもの, 仕様書から読み取ったものに分類できる.

一般的性質としては, 検査モデルをコンパイルすること自身が状態マシンの無矛盾性の検証になる. この他に, NuSMV の check_fms 等のコマンドを実行することでも検証した.

ソースコードから読み取った検査項目としては, 以下の二つが中心になる.

(1) asm_line状態マシンにデッドロックが無いこと
(2) 各ラベルに到達すること(到達可能性検査)

これらを検証することで, モデル変換自身の正しさも検証した. すなわち, このホワイトボックスレベルの検証は検査対象の検査と作成したモデル変換手順自身の検証と言う意味がある. よってこの検証は最初に実施した.

CTL 式を使用することで EF 検査が出来る. EF φ とは, “いずれ φ が真となる経路が存在する”という意味である. EF(asm_line=LABEL)という CTL 検査式は, “いずれ asm_line が LABEL に到達する経路が存在する”という意味となる. この EF 検査式を用いることで, 検査モデルの各ラベルへの到達可能性を検査することが出来た.

LTL 式では, AF または AG 検査しか行えない. AF φ とは, “その後のあらゆる経路で, いずれ φ が真となる”という意味である. AG φ とは, “その後のあらゆる経路で, 常に φ が真である”という意味である. したがって, LTL 式では, ある特定の条件化でのラベルへの到達可能性を検査することが出来ない.

アセンブラプログラムのモデル検査では, CTL 式が使用できることが重要であることがわかった.

6. 考察

アセンブラプログラムのモデル検査を実践することで, アセンブラプログラムのモデル検査の有用性が確認できた. アセンブラプログラムの検査では, CTL 式の使用が有効であることがわかった.

また, モデル変換スクリプトによる検査モデルの半自動生成を使用したモデル検査の実施により, アセンブラプログラムと検査モデルとの相互変換の可能性が確認できた.

ここで, 検査モデルは状態爆発を避けるために抽象化した論理レベルのモデルを使用する必要があることがわかった. 将来的に, モデ

ル検査ツールの進歩，計算機能力の向上により，レジスタモデルが利用可能となると考えられるが現状では論理レベルモデルの使用が妥当である．すなわち，検査は設計レベルの検査となることを意味する．ここから，検証されたアセンブラプログラムの開発を行う場合，アセンブラプログラム作成後にモデル検査を行うのではなく，設計段階で検査モデルを作成してこれを検査し，検査モデルからアセンブラプログラムを作成することが望ましいといえる．

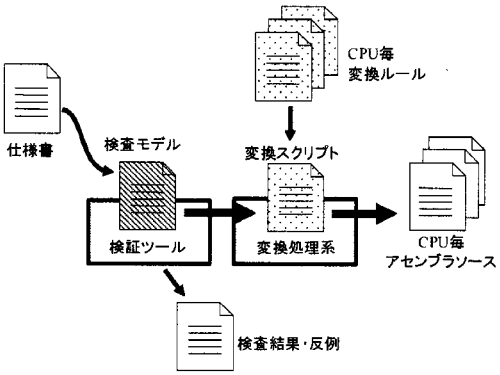

図 6 アセンブラプログラムの自動生成

7. まとめ

アセンブラプログラムのモデル検査の実践から，アセンブラプログラムのモデル検査に関する知見を得た．

アセンブラプログラムから検査モデルを半自動生成する有効性を示した．

本研究では，アセンブラプログラムのレジスタレベルでの検査が困難であることが判明し，設計レベルでの検査となった．形式的に検証されたアセンブラプログラムを開発するためには，実装後のアセンブラプログラムを検証するのではなく，形式的に検証されたモデルから，アセンブラプログラムを作成することが望ましい．これは，“イン-デザイン モデル検査”として水口らが提案する手順に相当する[4]．イン-デザイン モデル検査では，設計工程で仕様書に対するモデル検査を行うことを提案しているが，プログラム作成については言及が無い．本研究では，仕様書に対するモデル検査に加えて，さらに検証されたモデルからプログラムを自動生成することを提案する(図 6)．

アセンブラプログラムは，アセンブラ変換スクリプトにより検証済みの検査モデルから生成される．アセンブラ変換スクリプトは，CPU 毎の変換ルールにより対象 CPU 用のアセンブラプログラムを生成することが可能になる．

本研究では，アセンブラプログラムと検査モデルとの相互変換への可能性を示した．検査モデルからのアセンブラプログラムの自動生成が今後の課題となる．

本研究で使用したモデル変換スクリプトの生成する検査モデルは，アセンブラプログラムの行番号に依存する asm_line 変数を使用する．すなわち CPU に依存したモデルとなっている．検査モデルからアセンブラプログラムの自動生成を行う場合は，asm_line を使用しないモデルを作成する必要がある．

アセンブラプログラムの開発における，モデル検査の有効性から，さらに検査モデルからのアセンブラプログラムの自動生成のためには，以下の課題が考えられる．

検査モデルからアセンブラプログラムを生成するためには，対象となる CPU 依存部分を自動生成する仕組みが必要となる．

また，現状の検査モデルは，モデル生成の基となるアセンブラプログラムに依存した asm_line 変数がモデルの中心となっている．アセンブラプログラムを検査モデルから生成するためには，モデルが asm_line に依存しない形態になっている必要がある．

参考文献

[1] モデル検査ツール NuSMV:
http://nusmv.irst.itc.it/

[2] モデル検査ツール BLAST:
http://mtc.epfl.ch/software-tools/blast/

[3] モデル検査ツール BANDERA:
http://bandera.projects.cis.ksu.edu/

[4] 篠崎孝一, 水口大知, 石井健志, “組込みソフトウェア開発のイン-デザイン モデル検査 －設計工程における仕様書のモデル検査の提案－”, ソフトウェアテストシンポジウム 2004