モデル検査支援ツールとZIPCとの連携による設計検証の導入事例

富士ソフト株式会社　技術本部　次世代技術研究室　エキスパート
小池　隆

1. はじめに

近年、ソフトウェアの利用が日常生活の隅々にまで拡がり、ソフトウェアの障害が社会に与える影響は計り知れないものとなりました。ソフトウェアの信頼性向上は、安全・安心な社会を築く上で重要な課題となっています。

当社は、ITソリューションベンダーとして、組込み系から業務系まで幅広い分野におけるソフトウェアを開発しており、信頼のおけるソフトウェアを提供することは、企業としての社会的責任の一つでもあります。

そこで当社では、ソフトウェアの品質向上の施策の一環として、形式手法の一つであるモデル検査に取り組んでおります。

モデル検査への取り組みにおいて、ZIPCと連携することによってさらに高い効果を出すことができましたので、ここに紹介させて頂きます。

2. モデル検査とは

モデル検査とは、対象となるシステムに求められる性質を、そのシステムの可能なあらゆる振舞いにおいて満たしているかどうかを、自動的・網羅的に検証する技術です。

モデル検査の対象は状態遷移システムです。組込みシステムは状態遷移系としてモデル化されるため、モデル検査は組込みシステムの設計検証に多く使用されています。

図1に、モデル検査の仕組みを模式的に示しています。モデル検査を実行するのは、モデル検査器と呼ばれるソフトウェアです。モデル検査器には、状態遷移系としてモデル化されたシステムと、システムに対する制約条件が入力されます。モデル検査器は、システムが制約条件に違反した振舞いをしないかどうかを、状態空間の網羅的な探索によって検査します。制約違反が見つからなければ結果はOKです。もしも制約に違反する振舞いが見つかったら、結果はNGです。このとき、モデル検査器は制約違反に至るシーケンスを「反例」として出力します。反例は、制約違反が生じた原因を解析し、不具合を解消するのに役立ちます。

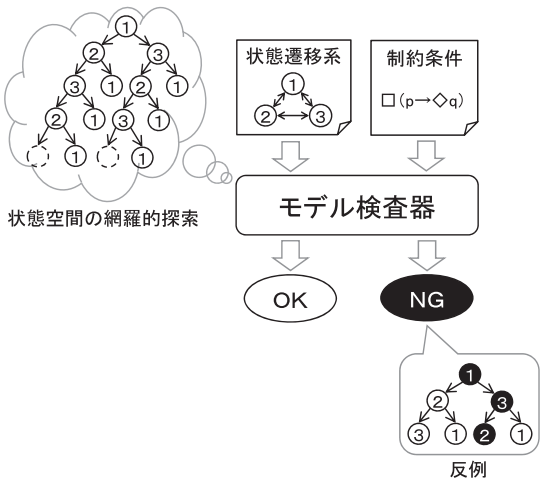

図1　モデル検査

3. モデル検査器SPIN

当社では、モデル検査器としてオープンソースのSPINを利用しています。

SPINでモデル検査をするには、状態遷移システムをPromela言語という専用の特殊な言語を用いて記述し、システムに求められる性質を線形時相論理（Linear Temporal Logic、以下、LTL）という論理式を用いて記述します。

単純な仕様の、架空のCDプレーヤをPromela言語で記述した例をリスト1に示します。構文はC言語風ですが、プログラムの意味を理解するには、実行可能性や非決定性といった独特の概念を知らなければなりません。

リスト1　Promelaプログラム

```
mtype = {NO_CD, PLAYING, WAITING}                /* 状態定義 */
mtype = {INS_CD, PLAY_BTN, STOP_BTN, EJECT}      /* イベント定義 */
mtype = {IN, OUT}                                /* CD の状態定義 */
mtype = {ON, OFF}                                /* モータの状態定義 */

active proctype SimpleCD() {

    mtype event;                                 /* イベント変数 */
    mtype state = NO_CD;                         /* 状態変数 */
    mtype cd = OUT;                              /* CD の状態変数 */
    mtype motor = OFF;                           /* モータの状態変数 */

    do
    :: true -> atomic {

        /* イベント発生 */
        if
        :: cd == OUT -> event = INS_CD;          /* CD 挿入 */
        :: true -> event = PLAY_BTN;             /* 再生ボタン */
        :: true -> event = STOP_BTN;             /* 停止ボタン */
        :: true -> event = EJECT;                /* イジェクト */
        fi;

        /* 状態遷移 */
        if
        :: event == INS_CD ->
            if
            :: state == NO_CD -> cd = IN; motor = ON; state = PLAYING;
            :: state == PLAYING -> assert(0);    /* 不可 */
            :: state == WAITING -> assert(0);    /* 不可 */
            fi;
        :: event == PLAY_BTN ->
            if
            :: state == NO_CD;                   /* 無視 */
            :: state == PLAYING;                 /* 無視 */
            :: state == WAITING -> motor = ON; state = PLAYING
            fi;
        :: event == STOP_BTN ->
            if
            :: state == NO_CD;                   /* 無視 */
            :: state == PLAYING -> motor = OFF; state = WAITING;
            :: state == WAITING;                 /* 無視 */
            fi;
        :: event == EJECT ->
            if
            :: state == NO_CD;                   /* 無視 */
            :: state == PLAYING -> cd = OUT; motor = OFF; state = NO_CD;
            :: state == WAITING -> cd = OUT; state = NO_CD;
            fi;
        fi;
    }
    od;
}
```

LTLは、命題論理に以下のような時相演算子を加えた論理体系です。

□ p　常に p が成り立つ
◇ p　いつか必ず p が成り立つ
pUq　q が成り立つまで p が成り立つ

常にリクエストに対していつか必ずレスポンスが返るという応答性は、以下のように記述されます。

□（*request* → ◇ *response*）

Promela言語もLTLも、一般のソフトウェア開発者には馴染みの薄いものです。そのため、SPINによるモデル検査を導入するには、これらを一から学習しなければなりません。このことが、開発現場へのモデル検査普及の大きな障壁となっています。

そこで当社では、Promela言語やLTLを知らなくてもモデル検査を利用できるようにし、プロジェクトへのモデル検査導入時の負荷をできるかぎり軽減しようと考え、モデル検査支援ツールを開発しました［1］。

4. モデル検査支援ツール

モデル検査支援ツールは、Microsoft Office Excel（以下、Excel）で作成した「状態遷移設計書」から、Promelaプログラムを自動生成します。

生成したPromelaプログラムを用いてSPINでモデル検査を実施することによって、状態遷移表を設計検証することができます。

4.1. 状態遷移設計書

状態遷移設計書のExcelファイルは、表1のような状態遷移表のシートと、後で説明する複数の付加的な表のシートで構成されます。

状態遷移表は、システムの振舞いを現在の状態とイベントとのマトリクスで規定します。状態とイベントの交点のセルの中には、上段には次に遷移する状態を、下段には実行するアクションを記述します。何もしないときは「／」で無視セルにし、状態とイベントの組合せがあり得ないときは「×」で不可セルにします。

表1　状態遷移表

	CDなし	再生中	停止中
CD挿入	再生中 CD受入 再生開始	×	×
再生ボタン	／	／	再生中 再生開始
停止ボタン	／	停止中 再生終了	／
イジェクト	／	CDなし CD排出	CDなし CD排出

状態遷移表には、システムに求められる性質は記述されません。通常のモデル検査では、システムに求められる性質はLTLを用いて記述することになりますが、LTLは難解です。

モデル検査支援ツールの状態遷移設計書では、表2のような「プロパティ表」を使用することによって、システムに求められる性質を簡単に記述することができます。

表2　プロパティ表

	CDなし	再生中	停止中
CD	OUT	IN	IN
モータ	OFF	ON	OFF

プロパティ表には、システムを構成する要素について、システムの各状態において取るべき値を制約条件として記述します。

表2の例では、構成要素として「CD」と「モータ」を挙げています。制約条件として、たとえば「再生中」状態における「CD」の値は「IN」（CDが入っていること）、「モータ」の値は「ON」（モータが回転していること）でなければならないことを示しています。

システムの構成要素の値は、アクションの実行によって変化します。そこで、状態遷移表に記載されたアクションの各々について、その実

行に伴ってシステムの構成要素の値がどう変化するかを「アクション表」に記述します。

アクション表には、そのアクションの実行前と実行後に成立していなければならない制約条件も記述することができます。「＝」は等号を、「←」は代入記号を表します。

表3　アクション表

	事前制約	処理内容	事後制約
再生開始	CD＝IN モータ＝OFF	モータ←ON	―
再生終了	CD＝IN モータ＝ON	モータ←OFF	―
CD受入	CD＝OUT	CD←IN	―
CD排出	CD＝IN	CD←OUT	―

表3のアクション表では、「再生開始」アクションの実行前には「CD」が「IN」で「モータ」は「OFF」でなければならないこと、そしてアクションを実行すると「モータ」は「ON」に変化することを示しています。

プロパティ表に記述された制約条件と、アクション表に記述された事前／事後制約は、Promelaプログラム中にassert文として出力されます。そのため、LTLを記述しなくても、モデル検査で制約違反を検出することができます。

アクションや遷移には、ガード条件を付与することができます。状態遷移表のセルで

[CDあり]再生中
[CDあり]再生開始

という記述は、ガード条件「CDあり」が成り立つときに、「再生開始」アクションを実行し、「再生中」状態に遷移することを表します。

ガード条件の判定内容は、表4の「条件判定表」に定義します。

表4　条件判定表

	判定内容
CDあり	CD＝IN
CDなし	CD＝OUT

さらに、システムの外部環境をモデル化するための「イベント表」もあります。外部環境を適切にモデル化すれば、不可セルへの非到達性も検査することができます。

詳細については参考文献を参照して下さい。

4.2. 反例解析支援

検査の結果、反例が出力された場合は、モデル検査支援ツールの反例解析支援機能を利用して不具合の原因を究明します。

図2は、表1から表3の状態遷移設計書をモデル検査して得られた反例を、反例解析支援機能を使用して表示したシーケンスです。

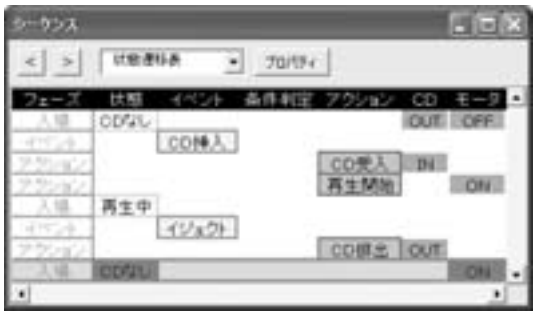

図2　反例シーケンス

図2のシーケンスを見ると、「CDなし」状態で「CD挿入」イベントが発生し、「再生中」状態に遷移したあと、「イジェクト」イベントが発生して「CDなし」状態に遷移した時点で、「モータ」プロパティの値が「ON」になっているという不具合が発生したことが分かります。

表2のプロパティ表では、「CDなし」状態での「モータ」プロパティの値は「OFF」でなければならないという制約を記述していますので、制約に違反しています。

「モータ」を「ON」から「OFF」に変更するのは、表3のアクション表によると「再生終了」アクションです。そこで、表1の状態遷移表で、「再生中」状態で「イジェクト」イベントが発生したときのアクションに「再生終了」を追加すればよいことが分かります。

モデル検査支援ツールでは、状態到達性や遷移網羅性など、あらかじめ用意したパターンに

基づいてLTL検査項目を自動生成し、検査を実施することもできます［2］。

5. プロジェクトへの導入

モデル検査支援ツールを用いて（実際にはモデル検査支援ツールの開発と並行して）、社内のソフトウェア開発プロジェクトにモデル検査を導入しました。

最初の年には、状態遷移表を作成しているチームが自分たちで検査するのではなく、別のチームで第三者検証的にモデル検査を実施しました。その大きな理由は、モデル検査支援ツールが開発途上だったことです。

当時は、自動生成されたPromelaプログラムの一部を、手動で変更しなければならないことがありました。その原因の一つは、状態遷移設計書に外部環境のモデルが含まれていなかったため、自動生成されたPromelaプログラムでは、本来あり得ないような反例（フォールスアラーム）が出力されてしまったことです。この問題は、外部環境のモデル化のための「イベント表」を追加したことによって解決しました。

もう一つの問題は、反例解析支援機能が提供できていなかったことです。SPINのGUIとしてXSPINというツールがありますが、XSPINで見られるのは、Promelaプログラムレベルでの反例シーケンスです。そのため、XSPINを使って反例を解析するには、自動生成されたPromelaプログラムの内容を理解していなければなりません。状態遷移表の不具合解析に本当に役立つのは、図2のような、状態遷移表レベルでのシーケンスです。

このような理由で、第三者検証的なモデル検査の導入をしましたが、いくつかの問題点も明らかになりました。

反例が出力されたとき、それが本当に設計上の不具合によるものなのか、それともフォールスアラームなのか、第三者には切り分けが困難な場合がありました。そのような場合には状態遷移表の作成者に判断してもらわなければなりません。フォールスアラームが続くと、現場の設計者はうんざりしてしまいます。

反例の原因が明らかにロジック的なものにあった場合には、設計者はモデル検査の威力を賞賛し、不具合を発見したことに感謝されます。しかし、不具合の原因がアクションの記述粒度の不統一のような些細なところにあると、重箱の隅をつつくようで、指摘された設計者はあまり快く思いません。次第に疎ましく思われてしまうこともありました。

そこで翌年からは、ツールの完成度が高まったこともあり、状態遷移表を作成するチームに自分たちでモデル検査を実施してもらうことにしました。

プロパティやアクションの粒度を統一し、外部環境を適切にモデル化し、システム内においても、システムとその外部においても整合性のとれた設計をすることの必要性を理解した上で、状態遷移設計書を作成するのです。フォールスアラームが発生しても、つまらない不具合が見つかっても、それは設計者が自分の責任として解決しなければなりません。導入時のハードルは少し高くなりますが、自分で実施することによってモデル検査についての理解が深まり、次第に作業効率が上がることも期待できます。

e-Learning教材やサンプルのモデルを整備し、簡単な教育を実施したあとは、後方支援に徹しました。その結果、導入プロジェクト数は伸び悩みましたが、一度導入したプロジェクトは、その後も継続してモデル検査を実施するようになりました。

3年目となる今年は、導入に成功したプロジェクトから他のプロジェクトへと、地道な水平展開によって、社内へのモデル検査の普及を推進しています。

6. 導入事例

当社において、2年余りの間にモデル検査を導入したプロジェクトは、表5のように多岐のドメインにわたります。

それぞれのプロジェクトにおいて、レビュー済みの状態遷移表を対象にしたモデル検査で、状態遷移表1つあたり数件の不具合を検出することができました。そのうちのいくつかを紹介します。

表5　モデル検査導入プロジェクト

項番	プロジェクト内容
1	携帯電話の内蔵アプリケーション
2	産業機械の表示モジュール
3	デジタルＴＶの通信モジュール
4	車載機器の内蔵アプリケーション
5	産業機械の通信モジュール
6	車載機器の通信制御モジュール
7	光学機器の制御モジュール
8	通信機器の監視制御モジュール
9	車載機器の通信モジュール
10	医療機器の入力モジュール
11	薬剤管理システム

項番6のプロジェクトでは、一度異常系のシーケンスに入ったあと、回復して正常系シーケンスに戻ったところで、「送信中」の状態でありながら「通信状態」プロパティの値が「非活性」であるという矛盾が、プロパティ制約違反によって見つかりました。

項番7のプロジェクトでは、モータを制御します。モータの回転方向を変える場合、一旦モータを停止し、完全に停止したことをセンサが検知した後でなければ回転させることができません。しかし、正回転中のモータを停止せずに逆回転してしまう可能性があることを、アクションの事前制約違反として検出しました。

項番9のプロジェクトでは、初期状態に戻ったときに、解放されていなければならないリソースが解放されていないことが、プロパティ制約違反として検出されました。

7. ZIPCとの連携

モデル検査の導入を希望するプロジェクトの中には、既にZIPCを利用しており、状態遷移表からCコードの生成まで実施しているプロジェクトがありました。

ZIPCで作成した状態遷移表とは別に、同じ内容で違うフォーマットの状態遷移表をモデル検査だけのために作成するというのは、無駄な作業です。手作業による再入力では、データ入力ミスが心配です。

そこで、ZIPCで作成した状態遷移表を、モデル検査支援ツールの状態遷移設計書にインポートするツールを作成しました。

インポートツールは、ZIPCで作成してXML形式で保存したSTMファイルを読み込み、状態遷移設計書のExcelファイルを作成します。

図3　ZIPC連携

インポートすると、状態遷移表のシートには、ZIPCの状態遷移表の内容が再現されます。ただし、NSチャート式の条件分岐はExcelでは表現できないため、独自の記法に変換されます。また、拡張階層化には対応できていないなど、制限事項があります。

プロパティ表とアクション表は、雛形のみが作成されます。プロパティ表には、状態名は列挙されますが、プロパティ名は列挙されず、制約条件の欄も空白となります。アクション表には、アクション名は列挙されますが、事前／事後制約と処理内容の欄は空白のままとなります。そのため、インポートした後、プロパティ表とアクション表を完成させてから、モデル検査をすることになります。

8. 導入成果

モデル検査支援ツールをZIPCと連携することによって、モデル検査で品質を確保された状態遷移表からC言語のコードを生成することができました。

モデル検査では、網羅的な検証を自動的に実行できるため、通常のテストでは見つけにくい不具合を検出することができました。これは、例外的な事象が重なったときにのみ生じ、実機で現象を再現させるのも困難なタイプの不具合でした。

ZIPCを使用せず、Excel等で状態遷移表を作成しているプロジェクトでは、設計の粒度が統一されていなかったり、曖昧さが残っていたりすることがあり、モデル検査のできる水準まで引き上げるのに一苦労することがあります。

ZIPCと連携した今回のプロジェクトでは、そのような問題は全くありませんでした。ZIPCでCコード生成することを前提に状態遷移表を作成しているため、もともと厳密なモデルを作成する習慣が身についていたことが理由として挙げられます。

ZIPCとの連携でモデル検査を導入するにあたり、当初、一つだけ心配がありました。それは、ZIPCを使ってCコード生成まで実施しているため、設計の粒度が細かく、モデル検査をすると状態爆発が起こりやすいのではないかということです。

状態爆発とは、検査において探索すべき状態数が膨大になり、コンピュータの記憶容量や処理時間の限界によって、検査しきれなくなってしまうことです。

ここでいう状態数とは、状態遷移表で定義された状態（抽象状態）の数ではありません。同じ抽象状態にあっても、そのときに保持している変数の値が異なれば、違う状態（内部状態）になります。変数の数が多かったり、それぞれの変数が取り得る値の範囲が広かったりすると、それらの値の組合せによって内部状態のバリエーションが爆発的に増加し、状態爆発が起こりやすくなります。

人手でコーディングすることを前提として作成された状態遷移表では、良くも悪しくも、「ほどほどに」記述されます。モデル検査の観点で見ると、粒度が細かすぎない点は良いのですが、粒度が恣意的で一貫性に欠けると反例が出やすくなってしまいます。

設計の粒度を細かすぎないようにしながら、一定の粒度で一貫性を持たせることが重要です。この難しい課題を、状態遷移表の作成規約によってクリアしました。

状態遷移表の作成規約は、もともとは可読性や保守性、移植性の向上といった、さまざまな観点から作成されたものです。しかし、そのうちのいくつかは、モデル検査をする上で非常に有用でしたので、その一部を紹介します。

●状態遷移表のアクションセルにはアクション関数コールを記述し、変数操作を直接書かないこと

実装レベルでの変数操作は、容易に状態爆発を引き起こします。そこで、アクションセルの中には変数操作を書かず、アクション関数コールを記述します。アクション関数の内容は、モデル検査用のアクション表では抽象化した変数操作を記述し、FNC設計書には実際の変数操作を記述します。それによって、モデル検査と実装コードで抽象度を変えて、1つの状態遷移表でモデル検査とCコード生成を両立させることができます。

●アクションセルで、for文、while文、do while文を使用しないこと

状態遷移表に、実装コードに近いレベルの記述をすると、粒度が細かすぎて、設計の再利用が難しくなります。モデル検査では状態爆発しやすくなります。

●アクション関数は単一の処理を実行すること

1つのアクション関数に複数の機能を持たせてしまうと、アクション関数によって粒度が統一されなくなってしまいます。また、本来は対称的であるべきアクション関数の対（獲得と解放、開始と終了など）の内容が、実際には対称的でなかったりすると、不整合が生じやすくなります。

9. 今後の課題

モデル検査支援ツールとZIPCとの連携によって、品質向上の成果が得られましたが、いくつかの課題も明らかになりました。

●未対応の機能

ZIPCには非常に多くの機能があり、拡張階層化をはじめとして、多様な状態遷移表を作成することができます。

モデル検査支援ツールの状態遷移設計書では、ZIPCのステート仮想フレームに相当する記述は可能ですが、その他の階層化には対応していません。

記憶型遷移やSTMコールなど、拡張階層化への対応の強化が望まれています。

●状態遷移表の記法の違い

現在のモデル検査支援ツールでは、状態遷移表をExcelで作成するため、ZIPCのようなNSチャート式の条件分岐は表現できないなど、表示上の制限があります。そのため独自の記法で対処していますが、ZIPCに慣れているユーザには違和感があるようです。

そこで、ZIPCと連携してモデル検査支援ツールを使用しているユーザからは、Excelを介さずに、ZIPCから直接モデル検査を実施できることが望まれています。

●プロパティ表とアクション表の生成

インポートツールでは、プロパティ表とアクション表は、雛形しか作成されません。

ZIPCプロジェクトには、状態遷移表以外に様々な設計書が含まれます。そのうちRAM設計書はプロパティ表に、FNC設計書はアクション表に類似した内容を含んでいます。そこで、RAM設計書とFNC設計書の内容を、プロパティ表とアクション表にインポートすることが考えられます。

しかし、Cコード生成のために記述されたFNC設計書やRAM設計書の内容は、粒度が細かすぎ、そのままではモデル検査に流用するのは適当ではないかもしれませんので、検討が必要です。

●ZIPCとのデータ交換

モデル検査で不具合が検出されると、試行錯誤をしながら状態遷移設計書を修正することになります。本当は、ZIPCで修正してインポートしてから再度モデル検査をすれば良いのですが、不具合の原因を探りながら修正するには、そのような手順は少々面倒です。そこでどうしても状態遷移設計書を修正してしまいます。

しかし、インポートはZIPCからモデル検査支援ツールへの一方向しかできませんので、状態遷移設計書を修正したら、ZIPCでも同じ修正をしなければなりません。これは二度手間ですし、修正を忘れると内容が一致しなくなってしまいます。

そこで、モデル検査支援ツールがZIPCのSTMファイルをエクスポートできるようになれば、利便性が増すと思います。

10. まとめ

社内で独自に開発したモデル検査支援ツールとZIPCを連携することにより、ソフトウェア開発現場の技術者が自分でモデル検査を実施し、品質の確保された状態遷移表からC言語のコードを自動生成することが可能になりました。

ZIPC導入済みのプロジェクトは、状態遷移表の品質のベースラインが高いため、スムーズにモデル検査を導入することができました。

モデル検査支援ツールとZIPCとの連携は、機能的にまだ不十分なところもあります。上流工程から下流工程までのツールチェーンを充実させ、それを現場のプラクティスとして普及・定着させることによって、より高品質なソフトウェアを高い生産性で開発できるようにしたいと思います。

参考文献

[1] 小池隆：状態遷移表に基づくモデル検査の支援環境，情報処理学会研究報告 Vol.2008 No.116 (EMB-10)

[2] 矢野恭平，小池隆：状態遷移表のモデル検査におけるLTL検証パターン，情報処理学会研究報告 Vol.2009 No.31 (SE-163)