日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

実行速度に依存するシステムの形式化と検証

水野 孝久, 西崎 真也

計算資源に制限がある状況においては，システムがプロセッサの実行速度に依存する場合がある．そのようなシステムを形式化する手法と，それに基づきモデル検査を適用する手法を提案する．

We discuss formlization of systems depending on execution speed of processors. We investigate how to apply model checkers to such formalization.

1 はじめに

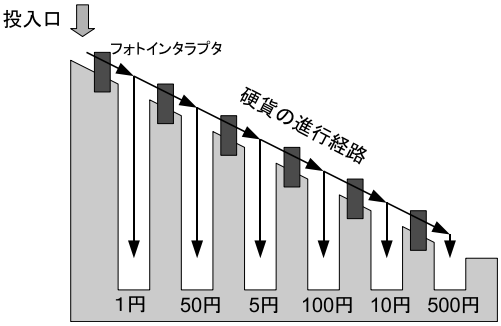
リアクティブシステムとは，あるシステムに外部環境から入力を与えると，それ応じて外部環境に出力をおこなうシステムのことである．リアクティブシステムは，入力から一定時間内に出力が行なわれることを期待されているが，仕様においてその時間が明確に指定されている場合もあれば，指定されていない場合もある．いわゆる，組み込みシステムはリアクティブシステムであることが多い．

リアクティブシステムにおける検証手法として最近，盛んに用いられているものがモデル検査器 (model checker) である．これは状態遷移図で記述されたモデルが，時相論理式で記述された条件を満たすかどうかを自動的に，かつ，網羅的に探索するソフトウェアである．UPPAAL [4] や NuSMV [1], [2] などがあるが，本論文では SPIN [3] を用いた．

また，検証の対象となるシステムとしては，貯金箱にマイコンにより制御される光センサおよび LED セグメントを付加したものとする．これについては次節で説明する．

2 電子化貯金箱

まず，ベースになったプラスチック製の貯金箱について説明する．この貯金箱は日本の硬貨のみが投入されるものとしている．投入口から硬貨が投入されると，それが傾斜のある通路をころがっていくように設計されている．通路には硬貨の大きさに応じた穴が空いていて，金種ごとに別々の場所に収納されるように設計されている．



穴は 500 円硬貨，100 円硬貨，50 円硬貨，10 円硬貨，5 円硬貨，1 円硬貨の 6 つが用意されていて，上から小さい大きさの硬貨から順に用意されている．この 6 種類の硬貨以外のもの，例えば，記念硬貨などは対象としない．

各々の穴の前にフォトインタラプタ (シャープ製

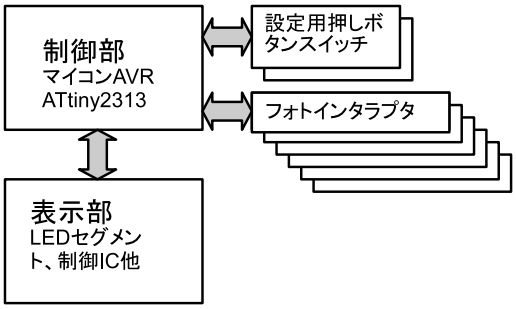
Formalization of Systems Depending on Execution Speed

Takahisa MIZUNO, 東京工業大学 大学院情報理工学研究科, Dept. of Computer Science, Tokyo Institute of Technology.

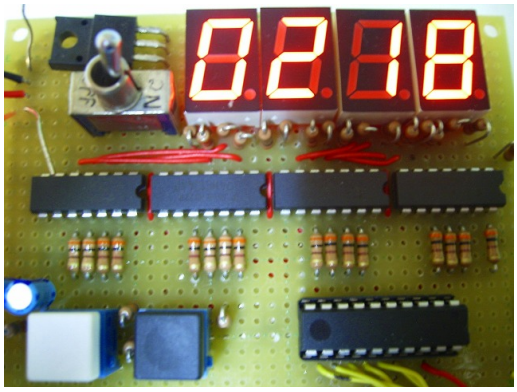
Shin-ya NISHIZAKI, 東京工業大学 大学院情報理工学研究科, Dept. of Computer Science, Tokyo Institute of Technology.

GP1A53HRJ00) を取り付けた．フォトインタラプタは LED と光センサから構成されていて，それらは向いあっている．物体がその二つの間を通過することにより物体の有無や通過を判定するセンサである．本論文では単に光センサと呼ぶこともある．6 つのフォトインタラプタの光センサの値の変化から，硬貨の投入，および，投入された硬貨が収納された場所がわかる．そのことから硬貨の金種がわかる．

本システムは、制御部、LED からなる表示部、フォトインタラプタ、設定用ボタンスイッチなどからなる。システム構成の概要図は以下のとおりである。



以下はシステムを実際に実装したものの写真である。



硬貨がフォトインタラプタに突入したときに，光センサからの信号は立ち下がり，光センサから硬貨が脱出したときに光センサからの信号が立ち上がる．たがって，本システムでは硬貨の通過の判定は，光センサからの信号の立ち上がりエッジを検出することによりおこなうことにした．硬貨の通過判定のアルゴリズムを疑似 C 言語コードで記述すると次のようになる．

```
int prevalue;
while(1){
  if(!Sensor && prevalue){
     硬貨通過時の処理
  }
  prevalue = Sensor;
}
```

Sensor には現在の光センサからの値が格納されており，Sensor の値を参照することにより現在のセンサの値を参照できるものとする．そして，繰り返し Sensor の値を参照することにより，光センサの値の変化を追跡することとする．条件式!Sensor&&prevalue によりセンサからの信号の立ち下がりを検出し，これにより硬貨の通過を同定する．信号の立ち上がりは今回の実装では利用していない．

2.1 システムの動作における問題点

実際に本システムを製作し動作させたところ次のような問題点を発見した．

2.1.0.1 問題点

硬貨の通過を検知できないときがある．

これに対して，製作中においては次のような変更により対処した．

2.1.0.2 対策

(1) 光センサの値の取得タイミングの変更

(2) プロセッサのクロック周波数の変更．1MHz であったものを 8MHz に変更した．

(1) について．当初は光センサの値を参照の都度読み出すようにしていた．これを前述の疑似 C 言語コードのように，一旦，光センサの値を変数 (下記のコードでは prevalue) の読み込んでから使用する．これはループの本体において，センサの値が変化する可能

性を踏まえた変更である．

これについて疑似Cコードで説明すると次のようになる．

```
int curvalue, prevalue;
while(1){
  curvalue = Sensor;
  if(!curvalue && prevalue){
    硬貨通過時の処理
  }
  prevalue = curvalue;
}
```

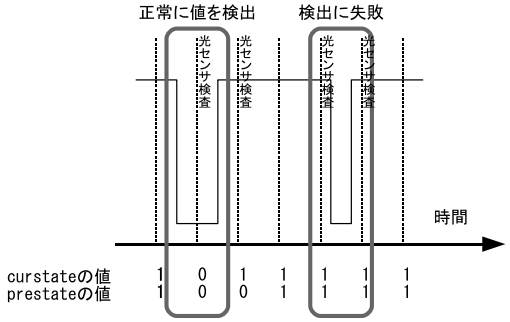
(2) について．これは，光センサの読み出し回数頻度が十分でないことを推測しておこなった処置である．

この二つの処置により，この問題点となった現象は起らなくなった．光センサの読み出し回数頻度の不足が (2) のような問題を引き起すことを，本論文ではモデル検査を用いて解明することを試みる．

3 実行時間への依存性の形式化

3.1 実行速度

本システムにおいて，プロセッサの実行と光センサの値の変化は独立している．硬貨が光センサを通過する時間に比べて，プロセッサのクロック周期が長い場合には光センサの値変化の検出を取りこぼすことがありうる．下記の図のような場合が考えられる．



3.2 プロセスの進行の一様化

本論文では，モデル検査システムとして SPIN Model Checker を用いる．モデルは Promela というモデル記述言語を用いて記述される．Promela では，手続き型言語の制御構造，通信チャネルを用いた送受信の機能，論理値や整数のような基本的なデータ構造，非決定的選択などが提供されている．制御構造や非決定的選択から時間の前後関係については表すことが間接的に可能である．

SPIN のようなモデル検査システムは，あらゆる実行経路について網羅的に探索をおこなう．本システムを SPIN に素朴に適用すると，あるプロセスだけが一方的に実行が進むような実行経路を探索したり，また一方で，あるプロセスだけの実行が進まないような実行経路を検査してしまう．しかし，本システムにおいて検証すべき実行経路はそのようなものを望んでいない．そのような病的な実行経路を排除することは，検査結果に対する理解のしやすさを向上することにつながる．

本論文では，各プロセスが一様に進行し，進行が一つのプロセスにかたよることはないという条件を課すことにより，モデル検査を効果的に行うことを提唱する．

プロセスの進行を一様化するために，各プロセスの進行を管理するためのプロセス scheduler を新たに一つ導入する．

プロセス scheduler の動作は以下の通りである．

while true do
 if プロセス scheduler 以外はすべて停止
  then 各プロセスの実行を再開させる

各プロセスは条件が整うと待機状態から実行が再開されるように記述されている．プロセス scheduler を導入により，各プロセスは scheduler からの再開の指示を待ち，再開の指示があった後，元来あった条件が満されたら実行が再開する．

うえの scheduler に対応する，

各プロセスの疑似コード，コードスケルトンのようなもの．

3.3 システムの抽象化と単純化

実際の電子化貯金箱を忠実に記述したモデルに対してモデル検査を行うと，状態数の爆発により現実的に検査が不可能になる．そのため何らかのシステムの抽象化や単純化をおこない，状態数の削減を行なわなければならない．

本論文でのモデル化においては次のような抽象化や単純化をおこなった．

時間　時間は，整数を用いて離散時間としてモデル化した．各プロセスはある一定の処理を繰り返す．各プロセスは各々固有の時間をもち，プロセスにおいて処理の繰り返しが行なわれる毎に時間を 0 に初期化する．したがって，各プロセスの時間は相異なりうる位数 p の巡回群 $\mathbf{Z}/p\mathbf{Z}$ であるといえる．

光センサの数　もともとの電子化貯金箱においては光センサは 6 つ搭載している．これをモデル化においては探索空間を狭くするために 2 つに減じた．

硬貨の投入のタイミング　実世界において，硬貨の投入は物理的制約の範囲で任意のタイミングで投入することができる．しかし，モデル化においては，モデルのもつ周期に応じて投入されるものとする．実世界に対応させて説明すると，硬貨がいずれかの穴におちた後に次の硬貨が投入され，穴に落ちる前の状態で硬貨の硬貨は貯金箱内に一枚だけであることを仮定している．

割り込み　電子化貯金箱において，タイマー割り込みを用いて LED セグメントの表示の処理を行なっている．モデル化においては，モデルを単純にするため，割り込みを省略している．

3.4　モデル化の概要

ここで紹介するモデルは 6 つのプロセスからなる．

主プロセス	`mainprocess`
センサー 1	`sense1`
センサー 2	`sense2`
硬貨	`coin`
スケジューラ	`scheduler`

これらのうち，スケジューラ以外の各プロセスは，実行状態変数を有している．

主プロセス	`mpstate`
センサー 1	`s1state`
センサー 2	`s2state`
硬貨	`cstate`
割り込み	`itstate`

各実行状態変数はともに，整数が格納される．実行状態変数は，アクションを行なうたびに 1 増大するようにモデルは定義されている．また，一連のアクションが終了すると 0 が格納されるので，実行状態変数の値は周期的になっている．各実行状態変数の周期は次のとおりである．

主プロセス	`MP_INTERVAL` + 3
センサー 1	`SENSE_END`
センサー 2	`SENSE_END`
硬貨	`COIN_INTERVAL`

また，これらのプロセスは各々，終了状態変数を有している．終了状態変数は，スケジューラ以外のプロセスの時刻の進行をあわせることに用いられる．終了状態変数は，**false** で初期化されており，アクションが 1 回実行されるたびに，**true** となる．終了状態変数がすべて **true** となったとき，スケジューラプロセスがすべての終了状態変数を再び **true** に初期化する．終了状態変数が **true** になったプロセスは他の終了状態変数がすべて **true** になるまで，待たさえる．これにより特定のプロセスのみ，時間の進行が進んでしまうような場合を抑制することができる．

主プロセス	`mpdone`
センサー 1	`s1done`
センサー 2	`s2done`
硬貨	`cdone`
割り込み	`itdone`

次に各プロセスの動作についてその概要を述べる．

プロセス mainprocess

mpstate の値	アクション
$0, \ldots$ $\ldots,$ MP_INTERVAL -1	mpstate を 1 増大
MP_INTERVAL	s1value,s2value の値を各々curs1value, curs2value に読み込む.
MP_INTERVAL $+1$	センサー 1 にコインが通過したことを検出したら coin1count を 1 増やす.
MP_INTERVAL $+2$	センサー 2 にコインが通過したことを検出したら coin2count を 1 増やす.
MP_INTERVAL $+3$	MP_INTERVAL で読み込んだ s1value, s2value の値を各々 pres1value, pres2value に格納する．実行状態変数 s1state を 0 に初期化する.

プロセス sense1

s1state の値	アクション
$0, \ldots$ $\ldots,$ SENSE_ON -1	何もしない
SENSE_ON	s1state を 1 増やす s1value を **true** に設定
SENSE_END	s1state を 0 に初期化 s2state を 1 に設定 s1value を **false** に設定 coin1count を 1 増やす
その他	s1state を 1 増やす

ただし，s1state の値にかかわらず s1done が **true** ならば，何もしない.

プロセス sense2

s2state の値	アクション
$0, \ldots$ $\ldots,$ SENSE_ON -1	何もしない
SENSE_ON	s1state を 1 増やす s2value を **true** に設定
SENSE_END	s2state を 0 に初期化 s2value を **false** に設定 coin2count を 1 増やす
その他	s2state を 1 増やす

プロセス scheduler

すべての終了状態変数が **true** になると，すべての終了状態変数を **false** に初期化する．他のプロセスが終了状態変数を **false** に設定することはない.

プロセス coin

cstate の値	アクション
0	cstate を 1 増加 s1state を 1 に設定
COIN_INTERVAL	cstate を 0 に初期化
その他	cstate を 1 増加させる.

4 モデル検査の適用と考察

このモデルにおいて検査すべき性質を

実際に通貨した硬貨とセンサによって計数された硬貨の差が 1 を超えない.

硬貨の枚数を数える変数 coin1count, coin1insert, coin2count, coin2insert は自然数ではなく，3 を法とする剰余類 ($\mathbf{Z}/3\mathbf{Z}$ の要素) とし，モデル検査の際の状態数爆発を回避した.

$$\neg \Diamond ((p_1 \wedge q_1) \wedge \ldots \wedge (p_6 \wedge q_6))$$

ただし，p_1,p_2,q_1,q_2 は各々次のように定義される．

p_1:coin1insert == 2

p_2:coin1insert == 0

p_3:coin1insert == 1

p_4:coin2insert == 2

p_5:coin2insert == 0

p_6:coin2insert == 1

q_1:coin1count == 0

q_2:coin1count == 1

q_3:coin1count == 2

q_4:coin2count == 0

q_5:coin2count == 1

q_6:coin2count == 2

定数の値を $c_1, \ldots, c_6$ という略称をふることとする．

c_1 MP_INTERVAL

c_2 COIN_INTERVAL

c_3 SENSE_ON

c_4 SENSE_END

$c_1, \ldots, c_6$ にさまざまな値を設定して，この LPL 式の検査をおこなった．

4.1 モデル検査と考察

モデルに対する検証は以下のようになる．モデルを記述した Promela コードは，付録 A で紹介されているものである．

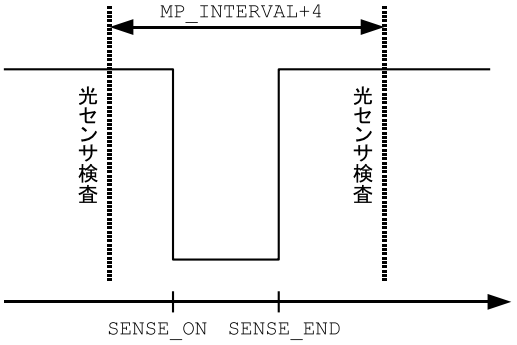
名称	c_1	c_2	c_3	c_4	結果
検証 1-1	1	15	6	11	不成立
検証 1-2	1	15	6	12	成立
検証 1-3	1	15	6	13	成立
検証 1-4	1	15	6	14	成立
検証 1-5	1	15	6	14	成立
検証 1-6	1	15	7	12	不成立
検証 1-7	1	15	7	13	成立
検証 1-8	1	15	7	14	成立
検証 1-9	1	15	8	13	不成立
検証 1-10	1	15	8	14	成立
検証 2-1	0	15	7	11	不成立
検証 2-2	0	15	7	12	成立
検証 2-3	0	15	7	13	成立
検証 2-4	0	15	7	14	成立
検証 2-5	2	15	7	13	不成立
検証 2-6	2	15	7	14	成立

考察

検証 1 は MP_INTERVAL を固定し，SENSE_ON と SENSE_END を変更している．

システムの実行速度が速いということは本モデルでは MP_INTERVAL が小さいということに対応する．これは主プロセスのメインループが一巡する時間間隔が短くなるということに他ならない．

SENSE_ON と SENSE_END の差が比較的小さいとき (≤ 5) においては，光センサがオンになっている時間間隔が短いため，検証 1-1，1-6，1-9 のように主プロセスが行うべき動作が失敗すると考えられる．要するに，実行速度が遅くなると動作が失敗するということがとらえられていることになっている．



検証 2 では，MP_INTERVAL の値を変更している．

MP_INTERVAL の値を小さくしたとき，主プロセスのメインループが一巡する時間間隔が短くなるため，検証 2-2 は SENSE_END と SENSE_ON の差が 5 であるが成立する．逆に MP_INTERVAL の値 2 とし，大きくしたとき，主プロセスのメインループが一巡する時間間隔が長くなるため，検証 2-5 は SENSE_END と SENSE_ON の差が 6 あるにもかかわらず，MP_INTERVAL＋4 以下であるから，不成立となる．

5 おわりに

本研究では，実行速度に依存するシステムの形式化の方法を提案し，その形式化に基づいたモデルに対してモデル検査を適用した．今回の適用事例から次のようなことがわかった．

- モデル検査において状態爆発を避けるためにモデルの単純化・を行なったが，その一方で，単純化・抽象化には限界があり，他の手法を適用することが必要である．
- 今回のモデル検査に対する適用結果は，実際に実装されたシステムの設計に活用できるほどのものではなかった．形式化されたモデルで得られる結果と，実際に実装されているシステムから得られる結果との十分な比較が望まれる．一つの例としては，実際に実装されている実行速度の調整に，形式化されたモデルから得られる結果を活用することが，考えられる．

SPIN を拡張して実時間システムの検証をおこなった研究として，Tripakis の研究[5] がある．これは時間 Büch オートマトンにもとづくものであり，Real-Time SPIN, Real-Time Promela として実装されている．本研究でおこなったモデル化は，Real-Time Spin や Uppaal などの時間オートマトンに基づくモデル検査系を用いればいくらか単純に記述できると考えられる．今回のモデル化では，s1state, s2state,mpstate,cstate の変数がクロック変数として用いられている．時間が同時に増加することは，プロセス schedule が管理している．時間オートマトンのクロック変数を Promela コードにより実現したものとなっている．今後，本研究でのモデル化をもとに Uppaal や Real-Time SPIN の上でモデル検査を行ない，本研究の成果と比較・検討することが重要と考えている．

参 考 文 献

[1] Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M.: NuSMV: a new symbolic model verifier, *Proceedings of International Conference on Computer-Aided Verification*, Lecture Notes in Computer Science, Vol. 1633, Springer, 1999, pp. 495–499.

[2] Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking, *Proceeding of International Conference on Computer-Aided Verification (CAV 2002)*, 2002.

[3] Holzmann, G. J.: *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley Professional, 2003.

[4] Pettersson, P. and Larsen., K. G.: UPPAAL2k, *Bulletin of the European Association for Theoretical Computer Science*, Vol. 70(2000), pp. 40–44.

[5] Tripakis, S. and Courcoubetis, C.: Extending Promela and Spin for Real Time, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, Vol. 1055, Springer, 1996, pp. 329–348.

A Promela により記述された電子化貯金箱のモデル

```
#define MP_INTERVAL 1
#define COIN_INTERVAL 15
#define SENSE_ON 15
#define SENSE_END 15

bool mpdone=true;
bool s1done=true;
bool s2done=true;
bool cdone =true;

int mpstate=0;
int s1state=0;
int s2state=0;
int cstate =0;


bool s1value=0;
bool s2value=0;

bool pres1value=0;
bool pres2value=0;

bool curs1value=0;
bool curs2value=0;
```

```
int coin1insert=0;
int coin2insert=0;

int coin1count=0;
int coin2count=0;

active proctype scheduler(){
        do
  ::true →
     if
     ::(mpdone==true && s1done==true &&
      s2done==true && cdone==true) →
      atomic{
       mpdone=false;
       s1done=false;
       s2done=false;
       cdone=false;
      }
     fi;
 od
}

active proctype mainprocess(){
 do
  ::true→
     if
     ::(mpdone==false)→
       if
       ::(mpstate==MP_INTERVAL)→
          atomic {
           curs1value=s1value;
           curs2value=s2value;
           mpstate=mpstate + 1;
          }
       ::(mpstate==MP_INTERVAL + 1) →
          mpstate=mpstate + 1;
          if
          ::(pres1value && !curs1value &&
            coin1count==2) →
            coin1count=0;
          ::(pres1value && !curs1value &&
            coin1count<2) →
            coin1count=coin1count + 1;
          ::else → skip;
          fi;
       ::(mpstate==MP_INTERVAL + 2) →
          mpstate=mpstate + 1;
          if
          ::(pres2value && !curs2value &&
            coin2count==2) →
            coin2count=0;
          ::(pres2value && !curs2value &&
            coin2count < 2) →
            coin2count=coin2count + 1;
          ::else → skip;
          fi;
       ::(mpstate==MP_INTERVAL + 3) →
          atomic{
           pres1value=curs1value;
           pres2value=curs2value;
           mpstate=0;
          }
       ::else →
          mpstate=mpstate+1;
       fi;
       mpdone=true;
     fi;
 od
}

active proctype coin(){
 do
  ::true →
     if
     ::(cdone==false) →
       if
       ::(cstate==0) →
          atomic{
           cstate=cstate + 1;
           s1state=1;
          }
       ::(cstate==COIN_INTERVAL) →
          cstate=0;
       ::else →
          cstate=cstate + 1;
       fi;
       cdone=true;
     fi;
 od
}

active proctype sense1(){
 do
  ::true →
     if
     ::(s1done==false) →
       if
       ::(s1state==0) → skip;
       ::(s1state==SENSE_ON) →
          atomic{
           s1state=s1state + 1;
           s1value=true;
          }
       ::(s1state==SENSE_END&&
           coin1insert<2)→
          atomic{
           s1state=0;
           s2state=1;
           s1value=false;
           coin1insert=coin1insert+1;
          }
```

```
      ::(s1state==SENSE_END&&
         coin1insert==2) →
        atomic{
         s1state=0;
         s2state=1;
         s1value=false;
         coin1insert=0;
        }
      ::else →
        s1state=s1state + 1;
      fi;
      s1done=true;
    fi;
 od
}
active proctype sense2(){
 do
  ::true →
    if
     ::(s2done==false) →
      if
       ::(s2state==0) → skip;
       ::(s2state==SENSE_ON) →
         atomic{
          s2state=s2state + 1;
          s2value=true;
         }
       ::(s2state==SENSE_END&&
          coin2insert<2) →
         atomic{
          s2state=0;
          s2value=false;
          coin2insert=coin2insert + 1;
         }
       ::(s2state==SENSE_END &&
          coin2insert==2) →
         atomic{
          s2state=0;
          s2value=false;
          coin2insert=0;
         }
       ::else →
          s2state=s2state + 1;
      fi;
      s2done=true;
    fi;
 od
}
```