修 士 論 文

動的更新が可能なソフトウェア開発手法の研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

阿部　友樹

2006 年 3 月

修 士 論 文

動的更新が可能なソフトウェア開発手法の研究

指導教官　片山卓也 教授

審査委員主査　片山卓也 教授
審査委員　二木厚吉 教授
審査委員　鈴木正人 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

410002 阿部　友樹

提出年月: 2006 年 2 月

概 要

本研究ではソフトウェアの動的な更新手法を実現するために，更新時に不具合が発生しないポイントを発見する手法の提案に取り組んだ。研究には状態遷移図と SPIN を利用した. 状態遷移図中には新たに状態の制約とサービスの情報を加え，それらを用いて更新可能であるか否かを判定する段階的な手法を提案した。

目 次

図 目 次

第1章　はじめに

1.1　背景と目的

近年，ユビキタス環境が整い始めている．ユビキタス環境とはユーザがいつでもどこでも計算機を意識することなくサービスが受けられる環境のことである．この環境においてサービスを提供するソフトウェアは永続的に動作することが求められる．

しかしこれらソフトウェアには停止することが避けられない状況がいくつか存在する．その１つとしてソフトウェアの更新作業があげられる．ソフトウェアの更新はバグの修正や機能の拡張，セキュリティの強化などを行う上で必要とされる作業である．現在の一般的な更新方法はシステムを一時終了する必要がある．この間はサービスを停止しなければならないので，サービスの品質低下に繋がる．また，あらゆるところに存在するシステムを手作業で更新するとなれば，莫大な人件費がかかる．これらの問題はユビキタス環境を整備する上で大きな問題となりうる．

そこで動的な更新方法が重要視される．ここで動的な更新とは，システムを終了させることなく，安全に新しいシステムに移行する技術のことである．動的な更新を行なうためには以下の手順が必要である．

1. 動作中のシステムから実行状態を抽出する．
2. 抽出した実行状態を新たなシステムに入力する．
3. 入力後，新たなシステムを正しいポイントから起動させる．
4. 古いシステムから新しいシステムへ動作を明け渡す．
5. 古いシステムを停止する．

この中で最も困難とされるのが手順３に書かれる「正しいポイント」の発見方法である．なぜなら「正しさ」の定義が明確ではない上に，システムの広大な状態空間の中から再開可能な[1]状態をを探し出すのは非常に困難であり，多大なコストがかかってしまうためである．

[1]再開可能とは動的にシステムの更新を行なっても問題が発生しないことを指す．再開可能と記述したのは，動的な更新を行なった際に一時停止は避けられないと考えたためである．以後再開可能と記述した時は，すべてこの意味を指しているものとする．

そこで本研究では状態遷移図を用いたアプローチで，この問題の解決に取り組んだ．再開可能な「正しいポイント」を状態遷移図で記される状態の特定の一部分に限定することで，有限な状態空間での探索が可能となる．「正しさ」については後の章で詳細に述べるが，大きくは次に示す 3 点に基準を置いた．

- 不正な値の操作を行なわない．
- デッドロックが発生しない．
- ユーザに不利益が生じない．

また有限な状態空間に限定することで，モデル検査ツール SPIN を利用することが可能となる．上記の基準の下で，SPIN を利用した再開可能なポイントの判定手法を提案した．

1.2 論文の構成

本稿の構成は下記に示すとおりである．

1 章 本研究の背景と目的について述べ，研究のアプローチを簡単に説明する．

2 章 研究で使用する関連技術に関して説明する．

3 章 本研究で提案した再開可能な状態の判定手法について述べる．この章の前半では研究を行なう上で置いた仮定や，再開可能と判定するための基準，状態遷移図の拡張などについて説明する．後半ではその前提の上で成り立つ段階的な判定手法について述べる．

4 章 本研究において使用する特徴的な Promela 記述方法について説明する．この章ではその記述方法について簡単に説明する．

5 章 本研究で用いた事例をベースにこの手法の使用方法について解説し，それと同時に手法の性能評価を行なう．

6 章 本研究をまとめ，今後の展望を述べる．

第2章　関連技術

この章では，本研究で使用する SPIN についての基礎知識，並びに SPIN に関連する Promela や線形時相論理についての説明を行なう．また本研究では状態遷移図を用いるアプローチをとり，状態遷移図の拡張を行なうため，ここでは一般的な状態遷移図について述べる．

2.1　SPIN

SPIN とは Simple Promela Interpreter の略称で，Bell 研究所の G.J.Holzmann らによって開発されたフリーのモデル検査ツールである．SPIN は 1980 年代に開発が開始され，今もなお継続的にバージョンアップされている．2002 年には社会的な功績から ACM System SoftWare Award を受賞した．

Promela(PROcess Meta LAnguage) とは SPIN が解釈できる仕様記述言語であり，並列動作をモデル化することができる．Promela は C 言語から多くの構文を取り入れている．本研究で使用する主要な記述に関しては後の節で述べる．

またモデル検査とは，システムの状態空間を網羅的に探索し，システムの性質を検査する技法である．対象を有限な状態空間で表現できるシステムに限定することで，状態空間の網羅的な探索を可能としている．複雑なシステムにおいては，検査したい性質のみを抜き出して抽象化する必要がある．

SPIN を用いて行なうことのできる検査は，デッドロックや飢餓状態などの並行システムにおいて起こりうる基本的な問題の発見の他に，後述する線形時相論理 LTL を用いたシステムの仕様検査などがあげられる．検査によってこれらの性質を満たさないケースを発見した場合，SPIN はエラーを出力する．

2.2　SPIN のツール構成

SPIN の構成を図 2.1 に示した．あるシステムを記述した Promela 文を検証器生成プロセスに与えると，検証器 pan を出力する．その検証器を ANSI の C 言語コンパイラにかけ，出力された実行ファイルを実行することによって検証を行なうことができる．エラーが出力された場合には，ガイドシミュレーション用の trail ファイルを生成する．ユーザは trail ファイルを用いて不具合の発生ポイントをシミュレートすることが出来る．

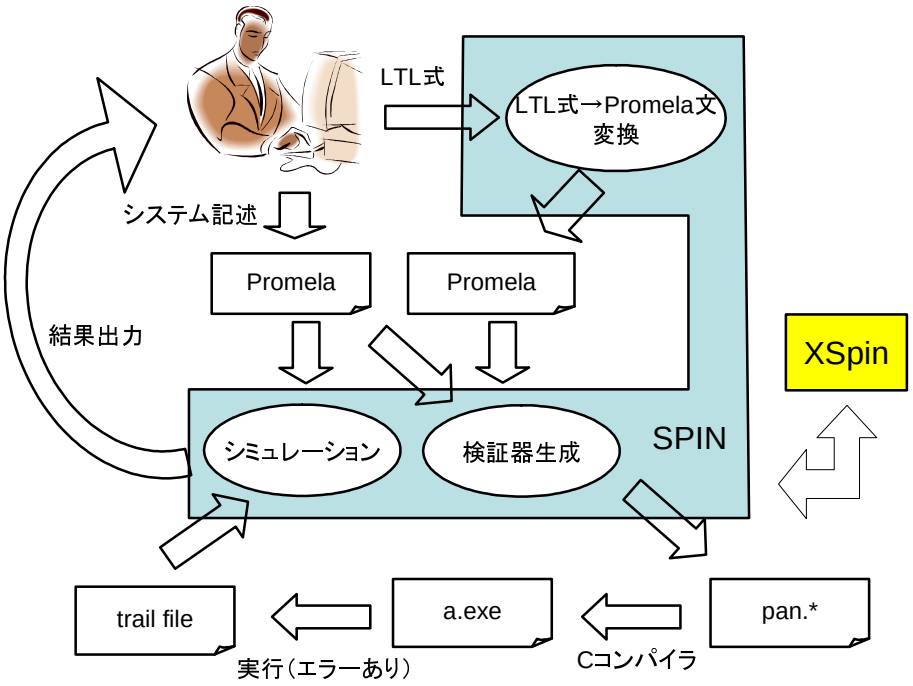

図 2.1: SPIN の概要

またユーザは SPIN に LTL 式を入力し，Promela 文に変換された LTL 記述を得ることができる．それをシステム記述の Promela 文と一緒に検証器生成プロセスに入れることで，システムの仕様を検査することができる．

本研究では XSpin というツールを使用する．XSpin とは SPIN をグラフィカルに表現する GUI フロントエンドである．XSpin はユーザから C 言語コンパイラの存在を隠蔽する画面インターフェースを提供する．

2.3 Promela 記述

この節では本研究で主に使用されている基本的な Promela 記述，atomic シーケンス，チャネル通信，ラベル，unless についての説明を行なう．

2.3.1 atomic シーケンス

atomic シーケンスは以下に示すように実行文を {} で囲み，その内部を実行する間は他のプロセスに対して排他的実行を行なう．しかし途中で実行文がブロック[1]した時は，制

[1]実行文が行なえなくなる状態になること

御が他のプロセスに移行される．ブロックした実行文が実行可能となると非決定的に遷移が戻る．

```
inline function(x, y){
    int temp;
    atomic{
            temp = x;
            x = y;
            y = temp;
    }
```

2.3.2 チャネル通信

チャネルとは並行動作するプロセス同士が相互に通信を行なうことのできる通信路である．通信路は大域的に定義する必要があり，以下の記述で書き表される．

```
chan チャネル名 = [サイズ] of {型名}
```

サイズとはチャネルが保持できるメッセージの数を示しており，サイズ分のメッセージが保持されている場合は，メッセージが受理されるまで送信がブロックされる．またサイズが 0 の時は同期通信を表しており，メッセージの送受信は同時に行なわれる．メッセージ送信は次のように記述する．

チャネル名!変数，変数，...

メッセージ受信は次のように記述する．

チャネル名?変数，変数，...

メッセージはバッファ先頭から取り出される．

また，通信に使われる型は Promela に用意されている，bit，bool，byte，short，int，mtype を使用することが出来る．ここで，mtype とはユーザ定義型のことである．

2.3.3 ラベル

ラベルは実行文に付加される．ラベルには end ラベル，progress ラベル，accept ラベルに加え，ユーザ定義ラベルがある．

end ラベルはデッドロックの検査時に正常な終了状態と，不正な終了状態を見分けるために使用される．このラベルは正常な終了状態の実行文に付加される．このラベルが張

られている実行文で停止する場合は，SPIN はそれを正当であると判断し，エラーを出さない．

progress ラベルは必ず一度は行なっているか否かを確かめたい命令文に付加される．この命令文の実行が行なわれないパスがある場合，SPIN はエラーを出力する．

accept ラベルは主に never claim で使用される．never claim は決して起こってはならないシステムの振舞いを検査する．accept ラベルを通過した場合，SPIN はエラーを出力する．

ユーザ定義ラベルとは，上記 3 つのラベル名以外のラベルで，goto 文の移行先や LTL 式でチェックしたい箇所に張ることができる．本研究では主にこのラベルを用いてシステムの仕様を検査する．

2.3.4 unless

unless は制御フロー文の一種で，命令文に優先度を付け加えるものである．しばしば do 文と共に用いられる．do 文とは繰り返し文のことを指す．使用例を以下に示した．

```
do
::g1 -> g2
::g3 -> g4
...
od unless {C}
```

この例では unless の後ろに記述されている実行文 C が実行可能でない限りは前半の do 文の実行を許し，C が真となると do 文がどの部分を実行していても，直ちに C を実行する動きをとる．ただし，C が真になるより前に break 文よって do 文内の処理が終了した場合は，C の実行は行なわれずに終了する．

2.4 LTL

LTL(Linear Time Temporal Logic) は時相論理の一種である．時相論理とは状態の遷移や時間の経過の観点からシステムの性質を記述するための論理体系である．時相論理は命題論理で使用される論理演算子 ($\neg$，$\vee$，$\wedge$) に加えて以下の時相演算子を用いて論理式を形成する．

- Xf　　(neXt)
  次の状態で f が成り立つ．
- Ff　　(Finally)
  いつかは f が成り立つ．

- Gf　　(Globally)
  いつも f が成り立つ.
- fUg　　(Until)
  g が成り立つまで f が常に成り立つ.

以下には本研究でよく用いる LTL 式の使用例を列挙した.

- invariance
  □ f：Gf
  いつも f が真である.
- eventually
  ◇ f：Ff
  いつかは f が真である.
- response
  □ (f → ◇ g)：G(f → Fg)
  f が真の時はいつも，いつか g が真となる.

2.5　状態遷移図

状態遷移図とはオブジェクトのライフサイクルを示した図である. オブジェクトはライフサイクル上のどれかの状態に属しており，イベントの発生により状態が移り変わる. これを状態が遷移すると言う. 状態は時間的な継続性を持つ. それに対しイベントによる遷移は瞬間的に行なわれる動作で時間を持たない. 瞬間的に行なわれる動作として，状態の入場動作，退場動作，そしてアクションがある. 入場動作とは，状態に入る際に必ず行なわれる動作である. 退場動作とは，状態から出て行く際に必ず行なわれる動作である. アクションとは，イベント遷移に付加された動作である. 継続的に行なう動作としては活動があげられる. 活動は状態内に滞在する間，継続的に実行される. 図 2.2 では状態遷移図の記述をまとめた.

本研究では状態遷移図の拡張を行い，これらの記述情報に新たな情報を付加する. この情報を用いて動的に更新することができるポイントの発見を行なう.

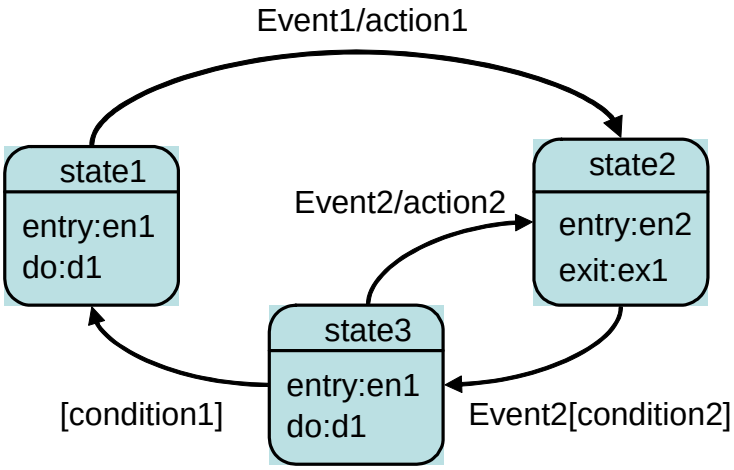
Event1/action1
state1
entry:en1
do:d1
state2
entry:en2
exit:ex1
Event2/action2
state3
entry:en1
do:d1
[condition1]
Event2[condition2]

図 2.2: 状態遷移図

第3章　再開可能な状態の段階的な判定手法

本研究では広大な状態空間から再開可能なポイントを限定するために，状態遷移図を利用した手法の提案を行なった．具体的には状態遷移図で描かれた状態内に滞在している時のみ再開可能であると限定した．状態内に滞在しているとは，入場動作や退場動作，アクションを行なっている最中ではなく，イベントの受理を待っているかまたは活動を行なっている状況のことである．この状況においてのみ更新手続きを許可した．

このようなアプローチで研究を行なう際に，いくつか仮定を置く必要がある．3.1 ではその仮定を説明する．また再開可能であると判定するための基準は現在は明確に定義されていない．その定義に関して 3.2 で述べる．3.3 では判定に用いる情報を状態遷移図に付加する方法について説明する．3.4 以降は状態遷移図に付加した情報を利用した段階的な判定方法の説明を行なう．

3.1　仮定

本手法を提案する際に，いくつか仮定を置いた．その仮定と仮定を置いた理由を以下に示した．

- システムは状態遷移図通りに実装されている．

  本研究は状態遷移図を利用したアプローチをとる．そのためシステムが状態遷移図と完全に対応づいていなければならない．状態遷移図とシステムとは今現在，完全に対応づいているとは言えない．本研究以降で必ずこの方法の提案を行なうべきであるが，現段階はその方法が確立されているものと仮定した．

- 実行中の旧システムからは現在の実行状態を抽出することができる．また新システムにその実行状態を入力し，正しいポイントからの再開がすることができる．

  システムの動的な更新を可能にするためには，実行中のシステムから実行状態を取り出すことが出来なくてはならない．また更新後に正しいポイントから再開するために，新システムには抽出した旧システムの実行状態を入力できる必要がある．この現実的な方法についても以後提案すべきであるが，これも実現されているものと仮定した．

- 新旧システムはそれぞれ単独では正しく動作するシステムである

  問題を更新時に限定するため，新旧システムを個々に稼動させた場合，どちらも不具合なく正常に動作するものと仮定した.

3.2 判定基準

更新後に再開可能であると判定する明確な基準は現在存在しない. 本研究を行なうには何らかの判定基準を定義しなければならない. そこで判定基準を定義するために，動的な更新を行なう上で起こりうる不具合を列挙し，それを考慮に入れた上で判定基準を制定した.

まずは動的更新において起こりうる不具合として考えられたものを以下に示す.

- システムの停止.
- システムの暴走.
- 新システムが旧システムの仕様を満足しない.
- 更新においてサービスの欠落や過剰提供が発生する.

前半の２項目はシステムのデッドロックや変数値を不正に操作したために起こる不具合である. 安全に新システムに移行するためにはこの不具合を必ず回避すべきである. 後半の２項目は利用者の観点から取り上げたもので，利用者に不利益を生じさせるタイプの不具合である. これら２つの不具合は更新後にクレームの対象となるので，回避すべきである.

本研究では上記４種類の不具合が発生しないポイントであれば再開可能であると判定する. 上記の不具合から洗い出された判定基準は以下の通りである.

1. システムが停止しない.
2. システムが暴走しない.
3. 不正な値の操作を行なわない.
4. 新システムが旧システムの仕様を満足する.
5. 更新時にサービスの欠落や過剰提供が起きない.

以上５点を基に手法の提案を行なった.

3.3 準備

提案した手法では既存の状態遷移図に記述可能な情報の他に，状態の制約に関する情報とサービスに関する情報を使用している．状態の制約とサービスに関しては後の項にて説明する．これらの情報を使用するために，状態遷移図を新たに拡張する必要がある．また判定基準にも登場している仕様に関しても明確に定義する必要がある．この節では状態遷移図の拡張記述と仕様の LTL 記述に関して説明を行なう．そして提案した手法を適用する前に新旧システムの状態遷移図にこれらの情報を全て書き加える．

3.3.1 状態遷移図の拡張

状態の制約

状態の制約とは，状態の入場動作が終わり，状態内の活動をしている最中かまたはイベントや条件の成立を待っている状況において，常に成り立つ条件式の集合である．状態の制約はシステム内の変数によって定義される．状態遷移図中には，状態の内部に [[constraint]] の形式で記述する．図 3.1 は状態の制約を記述した例である．

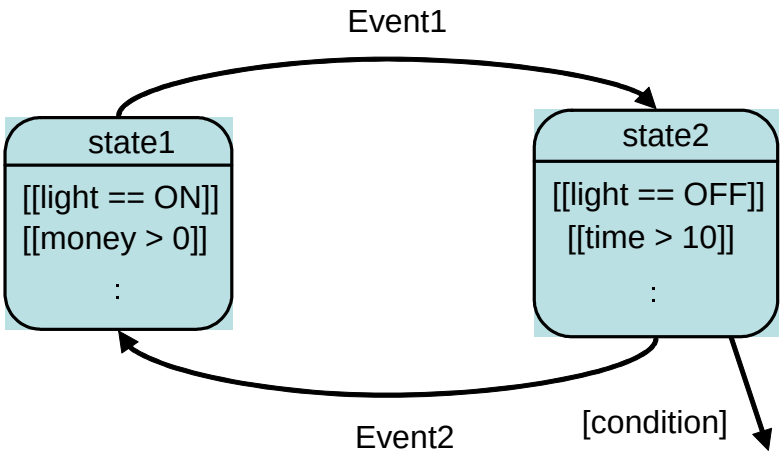

図 3.1: 制約の記述例

状態の制約は開発者が全てを取り決める．その制約が正しいかどうかの判定は後の章で述べることにする．

サービス

サービスとはユーザにとって有益な動作を指す．状態遷移図中には動作 (入場動作，活動，退場動作，アクション) の記述できるところであればどこにでも記述することが出来

る．状態遷移図には<service>の形式で記述する．図 3.2 はサービスの記述例である

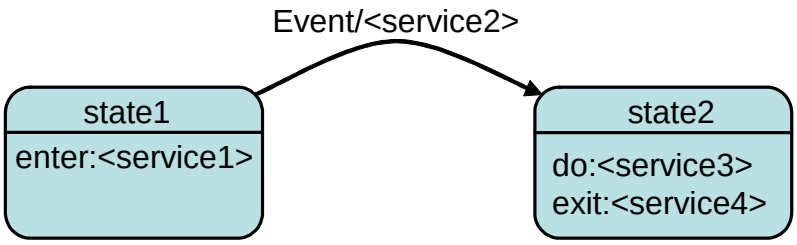

図 3.2: サービスの記述例

どの動作がサービスかという判断はシステムによって異なるため，全て開発者に委ねられる．サービスとして取り出した動作については必ず状態遷移図中に記述する．

3.3.2 仕様

仕様を LTL 式を用いて定義する．LTL 式は response の形式を用いる (2 章参照)．左側にはサービスの提供が発動するための条件文を記述する．条件文が複数ある場合には&&で結ぶ．条件部分はしばしば状態の制約と合致することがある．右側には左側の開始条件が満たされたことによって提供されるサービスを◇を用いて列挙する．提供するサービスが複数ある場合には&&で結ぶ．以下はサービスの記述形式と記述例を示す．

- 仕様の LTL 式記述

  □ ((サービス提供開始条件) → (◇サービス 1 && ◇サービス 2 && ...))

- 記述例 (電子レンジの例)

  – 扉が閉まっており，温め時間がセットされている状態でスタートボタンを押すと，マイクロ波が照射され，テーブルが回る．

  – □ ((door == OPEN && time >0 && STARTBUTTON) →
    ◇ service[STARTWAVE] == 1 && ◇ service[TURNTABLE] == 1))
    記述に登場する service[SERVICENAME] はサービスの数をカウントする変数配列である．この記述に関しての詳細は 4 章で述べる．

システム内の全ての仕様を LTL 式を用いて上記のように形式的に記述する．

3.4　段階的な再開可能なポイントの判定

これまでは手法を使用するための準備を行なってきたが，ここからはその準備した情報を利用して再開可能な状態を判定する手法について説明する．ここで，前節で述べた準備は全て完了しているものとする．

この手法では，はじめに旧システム中の各オブジェクト内の各状態に対し，新システム中の対応するオブジェクト内にある全ての状態を再開可能な状態の候補とする．そして以下に示す３つの検査を行なうことにより候補を絞る．図 3.3 は絞込みのイメージを示したもので，右側と左側の丸四角は新旧システムの対応するオブジェクトを示しており，点線の矢印は更新可能な状態の候補を示している．この図はステップをこなすことによって候補の数が絞られ，ステップを全てこなした候補を再開可能な状態と判定していることを示している。

最初のステップでは状態の制約を用いた候補状態の絞り込みを行なう．これにより不正な値をとる可能性がある候補状態を全て除去する．次に新旧の状態遷移図を結合して新システム中の候補状態に update 遷移というものを付加する．update 遷移とは旧システムから新システムに移行する時に通る遷移である．この遷移を付加したことでデッドロックが起きないかどうかを SPIN を用いて検査する．デッドロックが起きる場合には候補の状態から削除する．最後に LTL 式を利用したシステム仕様の充足性の検査を行なう．旧システムの仕様を満たしていない場合，また更新時にサービスが欠落，過剰提供する状態は候補から削除する．

この３つの検査を全てパスした時に候補状態が残っている場合には，その状態を再開可能状態と判定する．これらの段階的な検査はそれぞれが 3.2 で示した判定基準のいずれかの検査を担っている．以下では各ステップと対応する判定基準を示した．このステップを全てパスした時に判定基準が全て満たされる．以下では各検査ステップをまとめた．

STEP1 状態の制約による候補の絞り込み.

判定基準 3 を検査.

STEP2 システムのデッドロック検査.

判定基準１，２を検査.

STEP3 LTL 式による仕様検査.

判定基準４，５を検査.

この後の節ではこれらの段階的な手法の詳細を述べる．

3.5　STEP1：状態の制約による候補の絞り込み

旧システムの状態の制約と新システムの状態の制約を照合することで，不正な値の操作を行なう可能性を持つ状態からの再開を回避する．

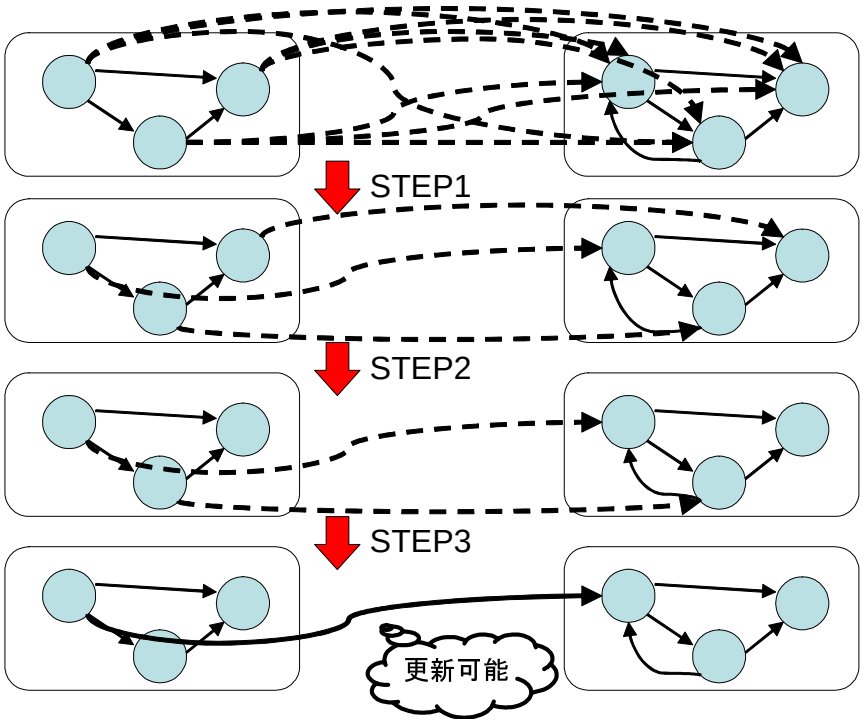

図 3.3: 絞り込みのイメージ

照合

以下の示す２つのうち，どちらにも属さない状態を不正に値を操作する可能性を持つ状態であると判定し，候補の状態群から外す.

- 状態の制約が完全に一致する.
- 状態の制約が完全には一致しないが，新システムの制約に旧システムの制約が包含される.

図 3.4 では正しいパターンと，不正なパターンの例を示している. パターン１では状態の制約が完全に一致しているため，正当であると判定する. パターン２では完全な一致はしていないが，新システムの制約 [[time >0]] に旧システムの制約 [[time >5]] が包含されているため，再開を行なっても矛盾のある再開にはならないので，正当であると判定する. パターン３では制約が一致も包含もされないため，不正であると判定し候補の状態から除外する.

新規変数の初期化

更新によって新たな変数が出来るケースがある. 新システムではその変数を制約に使用することがある. 旧システムではその変数自体が存在しないため，制約に用いることが出

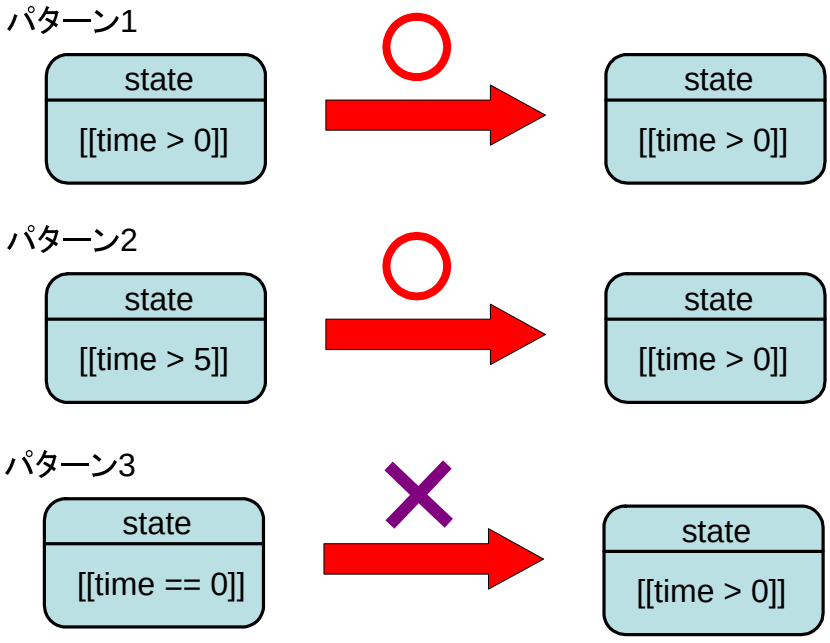

図 3.4: 整合性の判定パターン

来ない．そのままでは多くの状態が再開不可能と判定される．そこで新たに付け加わった変数に関しては，初期化時の値を旧制約に付け加えて判定する．図 3.5 は初期化の値を旧システムの状態の制約に加える例を示している．

この方法で旧システム内の全ての状態に対し，候補状態の絞込みを行なう．

3.6 状態遷移図の結合と update 遷移の付加

次の STEP2 の検査に入るに前に準備すべきことを述べる．まずは旧システムと新システムの状態遷移図を 1 つの状態遷移図に結合する．2 つの状態遷移図を同じ枠の中に記述する．最初はそれぞれが独立していて良い．更新後の状態遷移図の開始状態を示す記号は取り除く．次に update 遷移を付加する．update 遷移とは旧システムの状態からステップ 1 をパスした新システム中の候補状態へ向けられる遷移のことである．この遷移は旧システムと新システムを繋ぐ架け橋となる．update 遷移によって旧システムから新システムに移行されると，二度と旧システムには戻れず，新システムの実行のみを続ける．図 3.6 は状態遷移図の結合と update 遷移の付加を表現した図である．この図では，状態 s1 に対応する候補状態として状態 s1' が，状態 s3 に対応する候補状態とし状態 s3' があげられており，それぞれに update 遷移を付加してある．このように複数の update 遷移が存在する場合は，以後の検査において同時に複数の update 遷移をつけて検査を行なうのではなく，

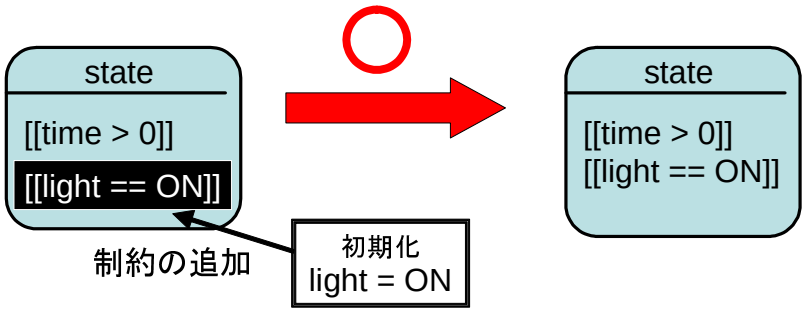

図 3.5: 新変数の初期化と制約の追加

1 本ずつ取り替えて検査を行なう.

3.7 STEP2：システムのデッドロック検査

前節で結合された状態遷移図を Promela 文で記述し，SPIN の到達性解析を用いてデッドロックが起きないかどうかを検査する．ここで旧システムと新システムはそれぞれ単体では正しく動作すると仮定しているので，この検査において SPIN がエラーを出力する場合は，更新時に不正な状態から再開していると判断することができる．この検査では更新によってシステムが停止，または暴走する状態からの再開を予防することができる．この検査でエラーが出力される状態は候補から削除する．

3.8 STEP3：LTL 式による仕様検査

このステップではユーザに不利益を生じさせないかどうかを検査する．この検査はユーザが人間かコンピュータかによって容認の範囲が変わってくるが，本研究ではユーザをコンピュータと仮定する．ユーザをコンピュータとした場合，ユーザはこれまでシステムに対して与えることのできていた刺激と，これまでに受けることのできていたサービスに関しては認知しているが，更新によって新たに加わった機能に関しては認知することができない．従って動的な更新が行われた時点で，ユーザは新たに加わったシステムへの刺激をシステムに与えることが出来ない．例えば更新によって新たに「ボタン A を押す」という刺激をシステムに与えることができるようになったとしよう．しかしユーザはその動作を知らないので，ボタン A を押すことができない．よってユーザは新たな刺激をシステムに与えられないのである．ユーザが求めるのは，これまでに存在していたシステムへの刺激のみで，これまでと同等かまたはそれ以上のサービスを受けられることである．

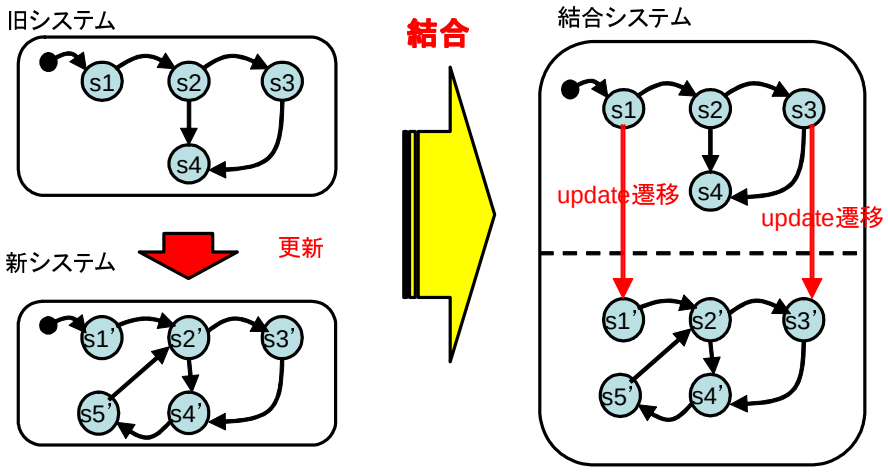

図 3.6: 状態遷移図の結合と update 遷移の付加

このような前提で検査する項目を 2 つ定義した．1 つ目は旧システムの仕様を新システムが補完しているかどうかを検査し，2 つ目は更新時にサービスの抜け落ちや重複提供が発生しないかどうかを検査する．この 2 つの検査をするための LTL 式をそれぞれ 2 つずつ，計 4 つの LTL 式で仕様の検査を行なう．以下の項では検査項目とその LTL 式を書き記した．

3.8.1 旧仕様の充足性検査

ユーザをコンピュータと仮定していることから，まずは新システムにおいて新たに加わえられたシステムへの刺激は使用不可能にする．すなわち，Promela 記述において新システムで新たに加わったユーザからシステムに向けてのメッセージを送信できないように，新たなメッセージ送信の実行文をコメントアウトすることに相当する．

次に update 遷移を通過した時に真となるフラグ変数 updateComp を用意する．この設定の下で新システムが旧システムの仕様を満足していることを確認する 2 つの LTL 式を提案した．その LTL 式を以下に示す．

1. □ ((サービス提供開始条件 && updateComp == true) →
   ◇提供されるサービス群))

   この式は旧システムの LTL 仕様記述の条件部に，さきほど定義した updateComp を付け加えた記述である．これはシステムが移行された後，新システムが旧システムの仕様を満たすかどうかを判定するための式である．update 遷移をつけずに単独で動く新システム上で仕様を確認することとの違いは，実行状態が移行されている点にある．

- 記述例 (電子レンジの例)
  ドアが閉まっていて温め時間がセットされている状態でスタートボタンを押すと，マイクロ波が放射され，テーブルが回る.(旧仕様)
- □ ((door == OPEN && time > 0 &&
  STARTBUTTON && updateComp == true) →
  ◇ service[STARTWAVE] == 1 && ◇ service[TURNTABLE) == 1)

2. □ ((サービス提供開始条件 && updateComp == true) → false)

   第 1 式ではシステムが仕様を満足するかどうかを確かめていたが，左側の条件部が常に偽の場合に SPIN は正しいと判定してしまう. この式は条件部分の式が成立するケースがあるかどうかを確かめるために右側部分を false に置き換え，SPIN によって検査する. エラーが検出される場合は左側の条件を満たすケースが存在していることを示すので，正当であると判定する. エラーが出ない場合には常に仕様の条件を満たさないことを意味しているので不正であると判定する.

   - 記述例 (上記と同様の例)
   - □ ((door == OPEN && time > 0 &&
     STARTBUTTON && updateComp == true) → false)

3.8.2 更新時におけるサービスの過不足検査

次に更新時点でサービスの欠落や重複提供が発生しないかどうかを検査する. 図 3.7 は前項の仕様検査を満たしているが，更新時にサービスの不足が起こる例である. この例では，新旧両システムはイベント E が起こるとサービス<s1>と<s2>を提供するという仕様を持っているが，状態 S2 で更新が起こり，S2' から再開されるとサービス<s2>の提供の抜け落ちが発生する.

図 3.8 は反対に更新時にサービスが重複提供される例である. この例でも新旧システムはイベント E が起こるとサービス<s1>，<s2>が提供されるという仕様を持っているが，状態 S2 で更新が起こり，S2' から再開されるとサービス<s2>の提供が重複して実行される. これらの検査を行なうために次の 2 つの LTL 式を用意した.

3. □ ((サービス提供開始条件) → (◇提供されるサービス群))

   この式は旧システムの LTL 式による仕様記述と全く同じである. 旧システム単独の仕様検査と異なる点は update 遷移が付加されている点である. update 遷移が付加たことで旧仕様のサービスが受けられなくなるかどうかを調べ，更新時のサービスの欠落を検査する.

   - 使用例 (前節と同様の例)

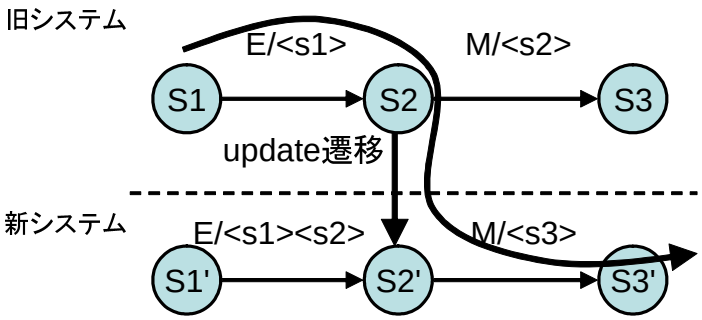

図 3.7: サービスの不足が発生する例

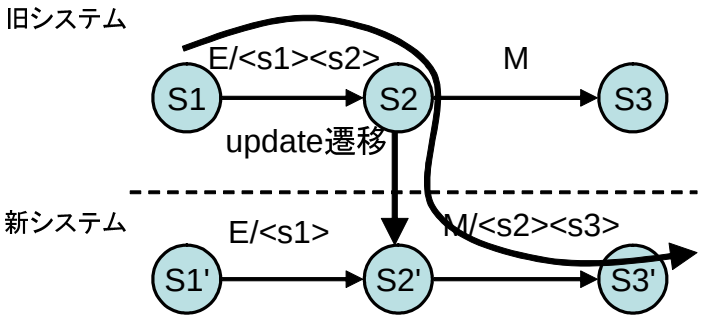

図 3.8: サービスの過剰提供が発生する例

- □ ((door == OPEN && time > 0 && STARTBUTTON) →
  ◇ service[STARTWAVE] == 1 && ◇ service[TURNTABLE) == 1)

4. □ (サービス 1 <= 1 && サービス 2 <= 1 && ...)

サービス 1 やサービス 2 に相当するのは，サービスのカウンタの値である．各種のサービスが行なわれるとそのサービスに相当する配列の値がインクリメントされる．この配列は仕様の条件部分が満たされた時に 0 にクリアされる．この変数は基本的に 0 か 1 の値しかとりえない．しかし更新が起こったことによってサービスが過剰提供される場合，カウンタがクリアされる前に重複してインクリメントされる．その時 SPIN はエラーを返す．

- 使用例 (前節と同じ例)
- □ (service[STARTWAVE] <= 1 && service[TURNTABLE] <= 1)

3.9 まとめ

これまでの 3 つの段階的な検査を行ない，全ての検査をパスした状態には 3.2 で定義した全ての判定基準を満たしているとして，動的更新が可能な状態であると判定する．以上が本研究で提案した手法である．

第4章　特徴的なPromela記述

SPIN を用いて検査を行なうためには，システムを Promela 文によって記述する必要がある．本研究では Promela 記述においていくつか特徴的な記述方法を利用した．本章ではその特徴的な記述に関して述べる．

4.1　状態

システムは状態遷移図通りに作成されていると仮定しているため，Promela 記述も状態遷移図通りに作成する必要がある．本研究ではシステム内の各状態はラベルを用いて定義している．以下にはラベルを使った状態の記述例を示す．この例では IDLE と OPEN がオブジェクト内の状態であり，それらがラベルで定義されている．

```
/*状態 IDLE*/
IDLE:
    statement1;
    statement2;
        :

/*状態 OPEN*/
OPEN:
    statement10;
    stetement11;
        :
        :
```

4.2　活動の記述

状態内ではイベントを受理するフェイズと，活動を行なうフェイズに分かれている．これはイベント受理と活動がそれぞれ別のメソッドで書かれていて，それらのメソッドが同時に実行されることがないためである．活動を行なっている最中はイベントの受理は待機され，イベントの受理を監視する間は活動を一時停止する．システムはこの２つのフェイズをスレッドスリープから次のスレッドスリープまでの微小時間で交互に行なう．この観

点から，活動を持つ状態のモデル化を次のように行なった．

```
STATE:
    /*イベント受理フェイズ*/
        :
INSTATE:
    /*活動フェイズ*/
        :
    goto STATE;
```

活動が存在する場合は，活動が始まる部分に状態名 STATE の先頭に”IN” をつけたラベルを貼る．活動の実行が終了するとイベント受理フェイズに戻る．このラベルを作成することにより，活動に関する仕様を LTL 式で記述可能となる．活動の存在しない状態には INSTATE を作る必要がなく，イベント受理フェイズで留まっていれば良い．以下の図は状態内の動作を示した概念図である．

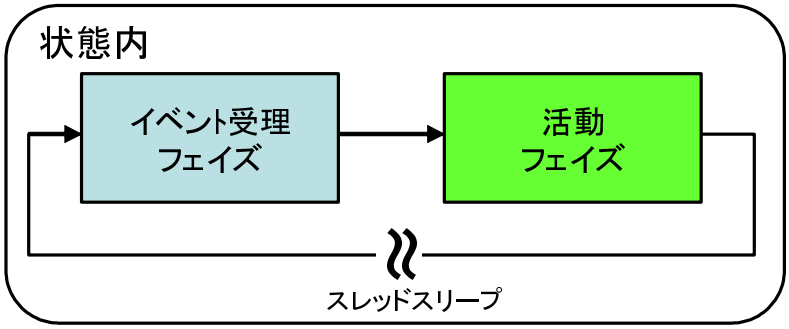

図 4.1: 状態内の動作

4.3 サービス

システム内の全てのサービスを define を用いて定義する．以下の項でサービスの提供した回数をカウントする配列を作成するので，サービスの定数値は重複しないように 0 番から順番に割り振る．サービスの記述順は特にこだわらない．サービスの基準は前章で定義したとおりで，ユーザに対して有益な動作のことである．どの動作をサービスとして定義するかは，使用するシステムによって異なるため，全て開発者に委ねられる．

```
/*サービスの定義*/
#define INCREASETIME 0     /*時間を１０秒増やす*/
#define SHOWDISPLAY 1    /*時間を表示する*/
#define CLEARDISPLAY 2   /*時間表示を消す*/
      :
```

4.3.1 サービスのカウンタ

サービスのカウンタとは，サービスの提供した回数をカウントする変数である．サービスは複数あるので，その値をカウントする変数を配列で定義する．この配列はシステム中の仕様の条件部分が成立された時に値を 0 にクリアする．またサービスが提供された時，そのサービスに対応した要素番号の配列の値をインクリメントする．要素番号は上記で定義したとおりである．サービスのカウンタ配列は service[SERVICENUM] と定義する．SERVICENUM はシステム中に存在するサービスの総数を示す．以下にはサービスのカウンタをクリアするメソッドを記述した．

```
/*サービスカウンタクリアメソッド*/
inline clearService(){
    int i;
    atomic{
         i = 0;
         do
         ::(i == SERVICENUM) -> break;
         ::else -> service[i] = false;
                   i = i + 1;
         od;
    }
}
```

この関数はシステム中のいずれかの仕様の条件を満たした時に呼び出され，全てのカウンタの値を 0 にクリアする．この関数があるためにカウンタの値は基本的には 0 と 1 となる．しかし前章で述べたように，更新時にサービスの重複提供が発生する場合にはその値が 2 となる．LTL 式による仕様検査では，この値を利用して重複提供が起こるかどうかを判定する．

4.4 連続的な動作

本研究で提案した手法の中に仕様を検査するステップが存在する．その仕様を SPIN を用いて検査する際に，Promela 文に何も処置を施さない場合にはエラーを出力すること

がある．これはユーザからの刺激を受けるオブジェクトが複数存在する場合に発生する．ユーザからの刺激を受けるオブジェクトが複数ある場合，SPIN は状態空間の網羅的な探索を行なうため，ある仕様におけるサービスを完結する前に他のオブジェクトに割り込みが入って，仕様のサービスが起こらないサイクルを発見する．

例えば図 4.2 のように，ボタンとレバーのついたシステムがあったとしよう．ボタンイベントとレバーイベントはそれぞれべつのオブジェクトが管轄している．ここでシステムにはボタンを押すとサービス 1 が提供され，レバーを引くとサービス 2 が提供されるという仕様があったとする．Promela 文に何も施さない場合は，ボタンが押されてからサービス 1 が提供されるまでの間に，レバーが引かれてサービス 2 を提供するという動作を繰り返すという意図しないサイクルまで探索してエラーを出力する．

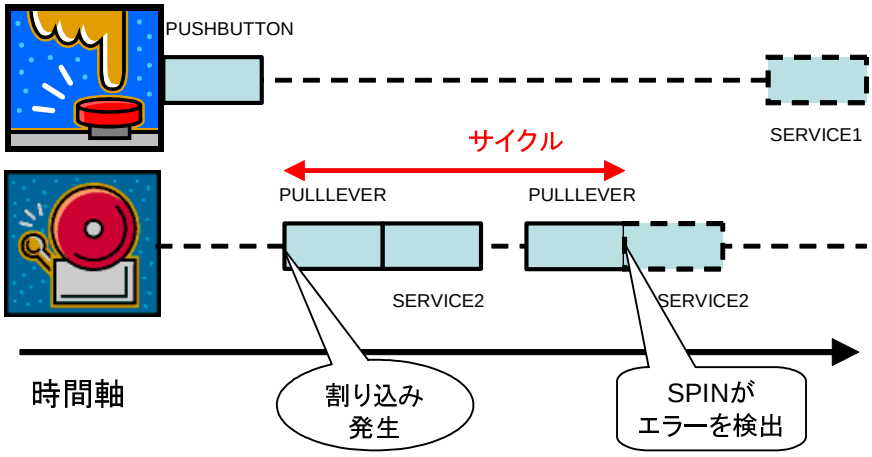

図 4.2: 意図しないエラーの検出

そこでこの意図しないエラー出力を回避するために以下の制約を設けた．

- 仕様の動作が完結するまでは他のオブジェクトに割り込まれない．

この制約を適用するために atomic シーケンスとイベント許可フラグを用いた．atomic シーケンスは実行の権利を他のオブジェクトに渡さずに連続的に実行を続けることの出来る記述である．イベント許可フラグとは仕様の動作が完了した時に真となるフラグで，ユーザからイベントを受けるオブジェクト全てに存在する．全てのフラグが真である時に，イベントの受理が可能となりユーザからのイベントを受け付ける．これは状態の活動部分にサービスが付加している仕様を検査する時に有効に働く．

これらを用いることで仕様が完結するまで他のオブジェクトに割り込まれないモデルを作ることができる．以下にはその記述例を示した．

```
proctype objectA(){
       :
STATE:
    if
    ::u_a?event ->
        a_b!message;
        :
    ::ok = false -> goto INSTATE;
    fi;

INSTATE:
    :
    ok = true;
    goto STATE;
}

proctype objectB(){
    :
STATE:
    if
    ::atomic{a_b?message ->
              statement;
                  :
}
```

この例ではオブジェクト A が状態 STATE にいる時，ユーザが event を出すと objectA がそのイベントを受け取り，objectB に向けて message を送る．objectB はメッセージを受け取ると atomic シーケンスに入り，他のオブジェクトに実行を渡さずに命令文を実行する．また objectA がイベントを受理せずに活動フェイズに入ったら，イベント許可フラグ ok を false にして活動に移る．活動が終了したら ok を真としてイベント受理フェイズに移る．この方法を用いることで，ユーザからのイベントを受理したり活動を行なう際に，他のオブジェクトに割り込まれずに実行を続けることが可能となる．ok のフラグの使用例は次の節で示す．

4.5 ユーザの動作

ユーザは前節で述べたイベント許可フラグが真の時にのみ，システムに対しイベントを起こすことが出来る．偽の間は待機している．そのモデルを unless を用いて以下のように記述した．

```
proctype User(){
HEAD:
ok == true;

    {if
    ::u_a!event1;
    ::u_a!event2;
    :
    fi} unless {ok == false};

    goto HEAD;
}
```

ユーザはイベント受理フラグ ok が真となったらイベントを送信できる．イベントの送信が行なわれたら unless 文を抜け出し，再びイベント許可フラグが真となるまで待機する．イベントを送信する前にシステム内のオブジェクトが活動フェイズに移ったら ok は偽となり，その瞬間 unless 文を抜け出して再びイベント許可フラグが真になるまで待機する．

4.6 update 遷移

update 遷移は結合した状態遷移図同士を繋ぐ遷移である．旧システムから新システムに向けて遷移する．遷移自体は他のイベント遷移と全く変わらない．

第5章　手法の性能評価

本章では3章で提案した手法の性能を評価する. しかし現時点でこの手法を検証できる状態遷移図通りに実装されたシステムが存在しないので，ここでは提案した手法の正当性と有効性の評価を議論する. そこで本章では本研究で使用した事例に対して提案した段階的な検査手法を適用し，実践的に動作を観察しながら評価を行なう.

5.1　事例の説明

本研究では簡単な電子レンジの事例を作成し，それを用いて性能評価を行なった. この節では電子レンジの仕様や更新内容について説明する.

5.1.1　旧システムの状態遷移図

旧バージョンの電子レンジは，図 5.1 に示される状態遷移図通りに実装されているものとする.

5.1.2　旧システムの仕様

電子レンジにはTIMEボタン，STARTボタン，扉，ディスプレイ，マグネトロン，テーブルが備わっている. ユーザがシステムに対して与えることのできる刺激は以下の通りである.

- TIMEボタンを押す.
- STARTボタンを押す.
- 扉を開ける.
- 扉を閉める.

システムはこれらの刺激によって何らかの条件を満たした時にサービスを提供する. 本研究ではこれを仕様と呼ぶ. 以下には事例における仕様とそのLTL記述を示した。

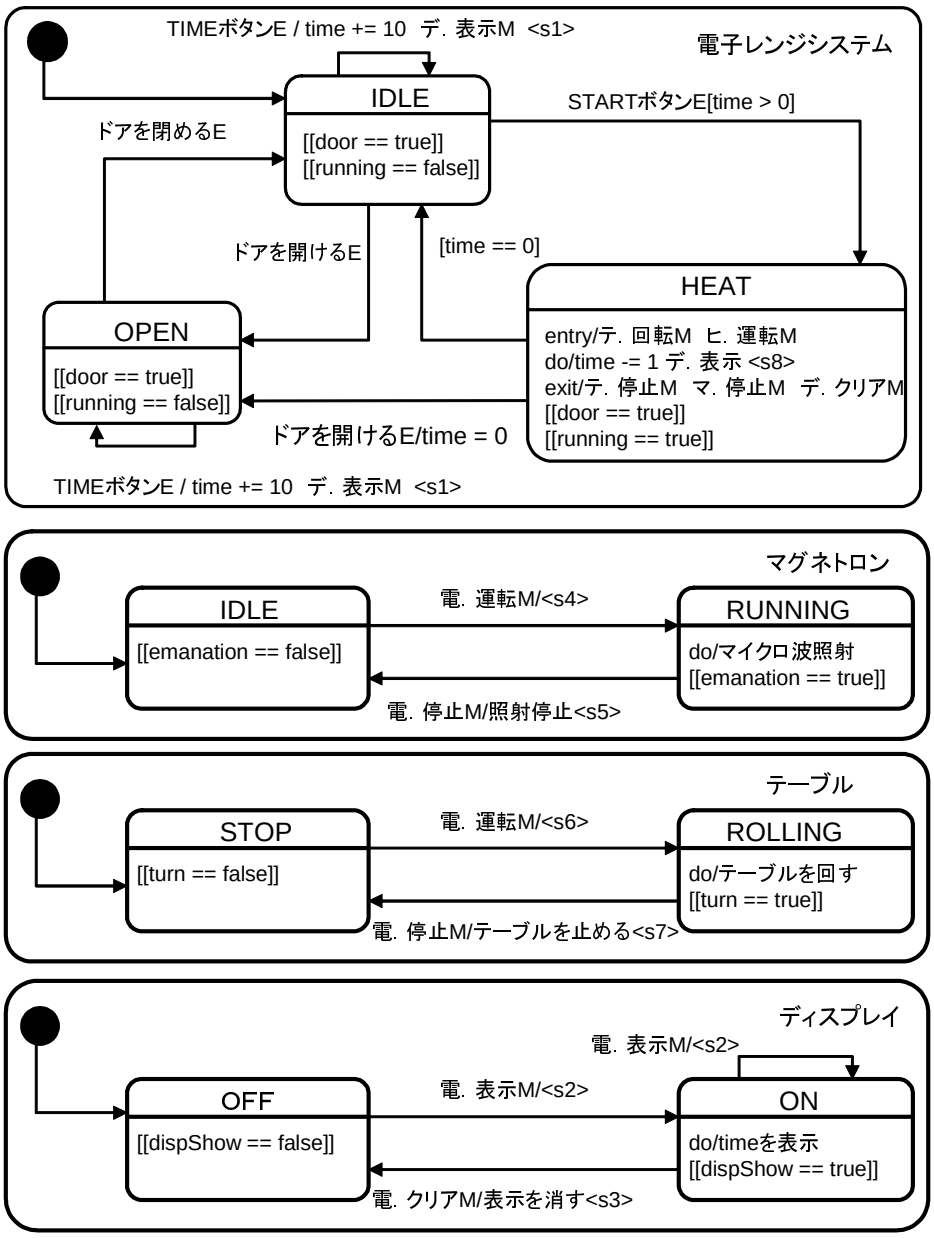

図 5.1: 旧電子レンジの状態遷移図

- 電子レンジ停止中に TIME ボタンを押すと，温め時間が１０秒増え，時間を表示する.

  □ ((e1 && c1 && c2) → (◇ s1 && ◇ s2))

```
#define e1 TIMEBUTTON
#define c1 running == false
#define c2 time < 20
#define s1 service[INCREASETIME] == 1
#define s2 service[SHOWDISPLAY] == 1
```

- 電子レンジ停止中に，扉が閉まっており，温め時間がセットされている時に START ボタンを押すと，マイクロ波が照射され，テーブルが回転する.

  □ ((e1 && c1 && c2 && c3) → (◇ s1 && ◇ s2))

```
#define e1 STARTBUTTON
#define c1 door == true
#define c2 running == false
#define c3 time > 0
#define s1 service[STARTWAVE] == 1
#define s2 service[STARTROLLING] == 1
```

- 電子レンジ稼動中に温め時間が 0 になると，マイクロ波の照射を停止し，テーブルを止め，ディスプレイを消す.

  □ ((e1 && c1 && c2) → (◇ s1 && ◇ s2 && ◇ s3))

```
#define e1 OPENDOOR
#define c1 door == true
#define c2 running == true
#define s1 service[STOPWAVE] == 1
#define s2 service[STOPROLLING] == 1
#define s3 service[CLEARDISPLAY] == 1
```

- 電子レンジ稼動中に扉を開けると，マイクロ波の照射を停止し，テーブルを止め，ディスプレイを消す.

  □ ((e1 && c1 && c2) → (◇ s1 && ◇ s2 && ◇ s3))

```
#define e1 OPENDOOR
#define c1 door == true
#define c2 running == true
#define s1 service[STOPWAVE] == 1
#define s2 service[STOPROLLING] == 1
```

- 電子レンジ稼動中は温め時間のカウントダウンを行い，その都度時間を表示する．
  □ ((c1 && c2 && c3 && l1) → (◇s1 && ◇s2))

```
#define c1 door == true
#define c2 running == true
#define c3 time > 0
#define l1 ElectricOven@HEAT
#define s1 service[COUNTDOWN] == 1
#define s2 service[SHOWDISPLAY] == 1
```

いくつか変数が登場したので，変数の定義を以下に示す．

```
#define running (emanation == true && turn == true && dispShow == true)

bool emanation;    /*マイクロ波放射状態　放射中：true　停止中：false*/
bool turn;       /*回転テーブルの状態　回転中：true　停止中：false*/
bool dispShow;  /*ディスプレイの表示状態　表示中：true　消灯中：false*/
int time;        /*温め時間*/
bool door;       /*ドアの状態　閉：true　開：false*/
```

5.1.3　サービスの定義

旧バージョンの電子レンジにおける全てのサービスを次のように定義した．

```
#define INCREASETIME 0    /*時間を１０秒増やす*/
#define SHOWDISPLAY  1    /*時間を表示する*/
#define CLEARDISPLAY 2    /*時間表示を消す*/
#define STARTWAVE    3    /*マイクロ波の放射を開始する*/
#define STOPWAVE     4    /*マイクロ波の放射を停止する*/
#define STARTROLLING 5    /*テーブルを回転させる*/
#define STOPROLLING  6    /*テーブルを停止させる*/
#define COUNTDOWN    7    /*温め時間を１秒減らす*/
```

5.1.4 更新内容

これまで説明してきた旧システムに対し，次の更新を行なう．

- モード切替ボタンを追加する．

  電子レンジには新たにオーブン機能が付け加わる．このボタンを押すと，レンジモードとオーブンモードを切り替えることが出来る．このボタンが押せるのは電子レンジもしくはオーブン停止時のみである．

- オーブン機能を追加する．

  モードがオーブンにセットされていて，扉が閉まっており，温め時間がセットさている状態で START ボタンを押すとオーブンが稼動する．

- 電力切替ボタンを追加する．

  レンジのワット数を切り替えることが出来る．ワット数は６００ｗと５００ｗの２つである．

- ライトを追加する．

  レンジまたはオーブンが稼動している時に点灯する．

これらの更新を加えた新バージョンの電子レンジの状態遷移図を図 5.2 と図 5.3 に示す．更新によって新たに加わった変数と，その初期化の値は以下の通りである．

```
#define running ((oven == true || emernation == true)
        && table == true && dispShow == true &&light == true)
bool light;  /*ライトの状態　点灯：true　消灯：false*/
  初期化: light = false;
  ※更新時点でライトは点灯していないため．
int watt;    /*電力の値*/
  初期化: watt = 600;
  ※もともと 600 wだったが更新前は明示する必要がなかった
bool oven;   /*オーブンの稼動状態　稼動中：true　停止中：false*/
  初期化: oven = false;
  ※更新前はオーブン機能が存在しなかったため
bool mode;   /*モード　電子レンジ：true　オーブン：false*/
  初期化: mode = true;
  ※更新前は常にモードは電子レンジであった
```

更新によってユーザがシステムに対して新たに与えることが可能となった刺激を以下に示す．

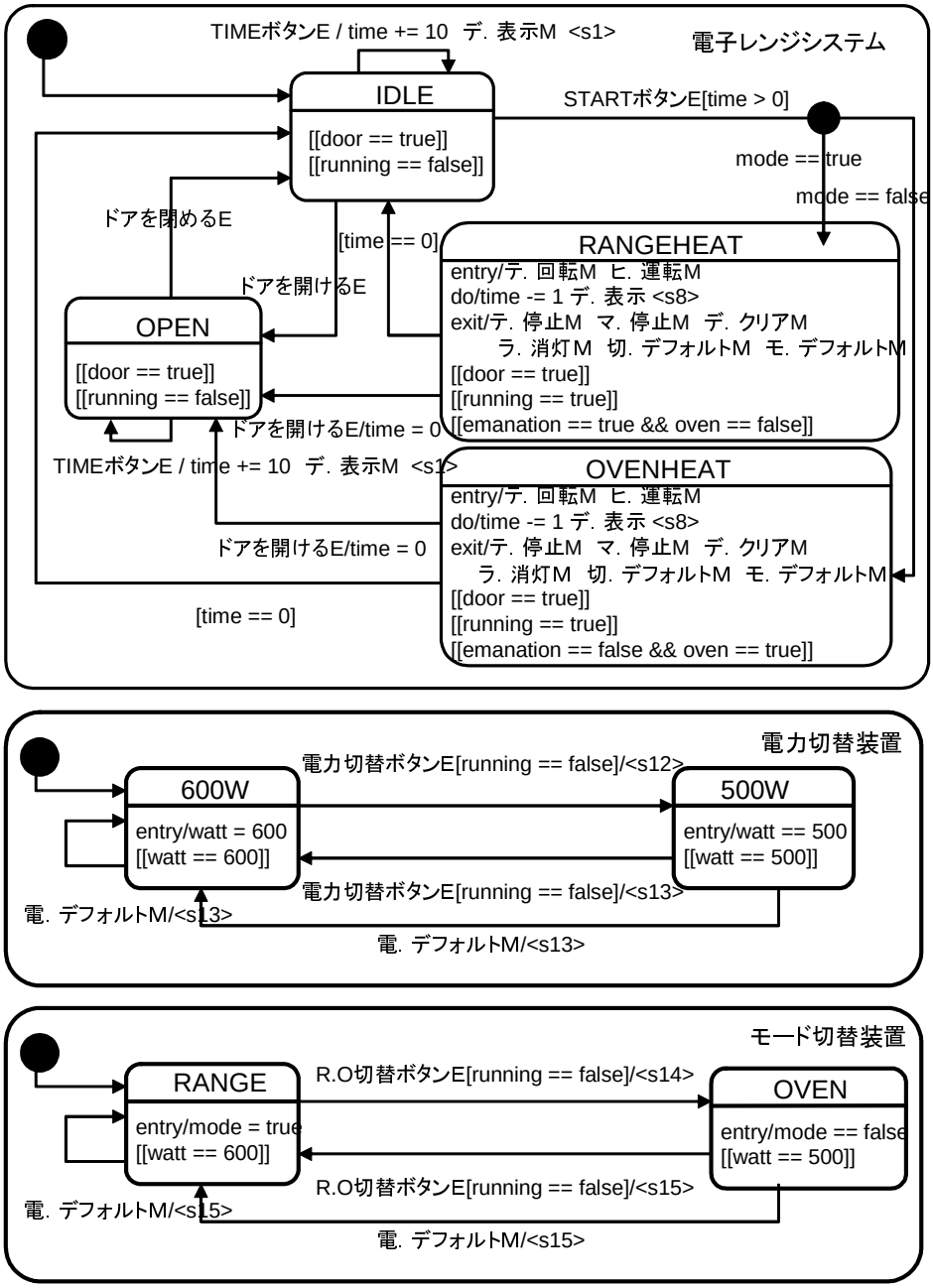

図 5.2: 新電子レンジの状態遷移図 1

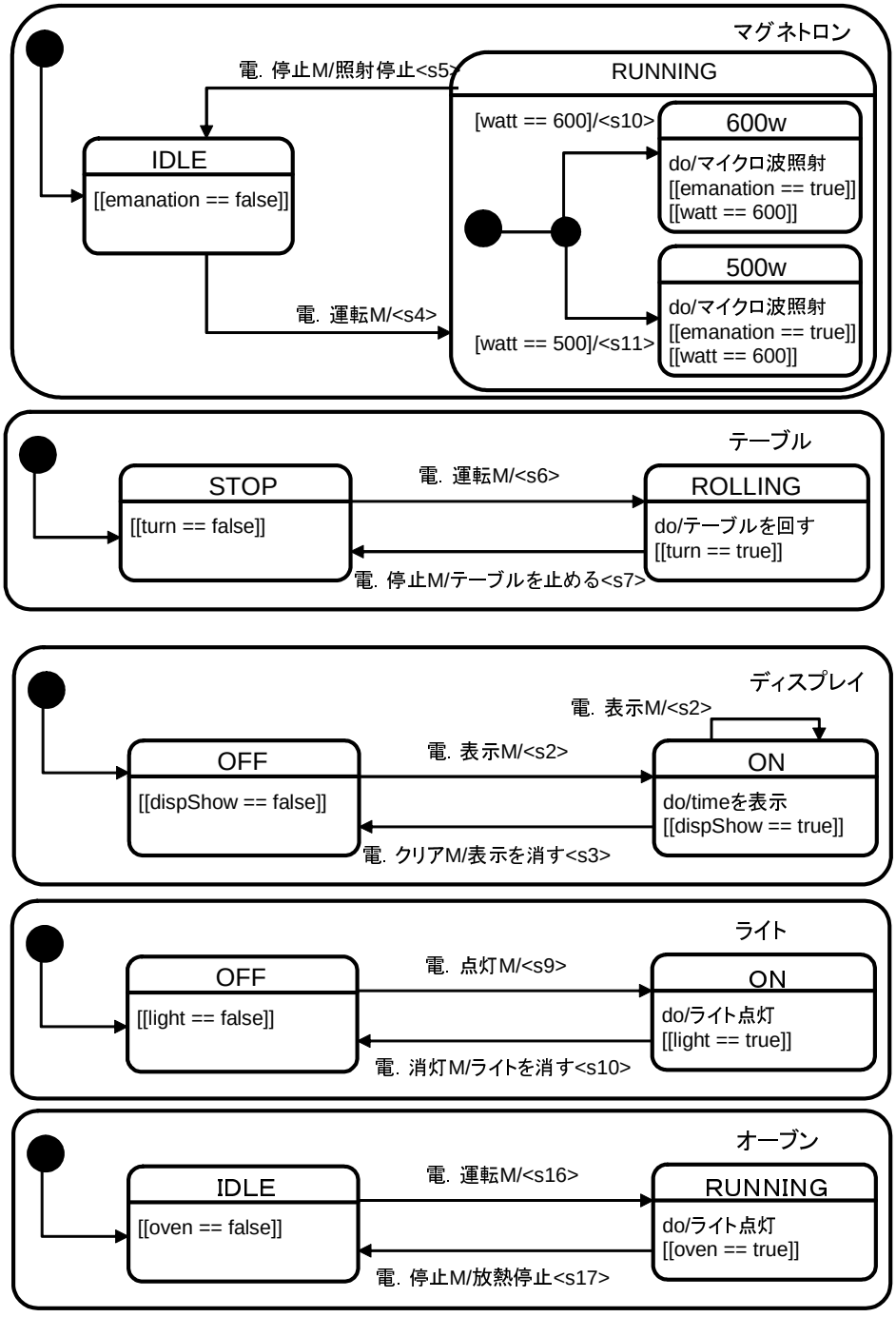

図 5.3: 新電子レンジの状態遷移図 2

- モード切替ボタンを押す.
- 電力切替ボタンを押す.

仮定より，新旧システムはそれ自体単独では正しく動作する必要があるので，事例の旧システムでこれらの仕様が満たされているか検証し，正しく満たしていることを確認した.

この後の節では3章で述べた段階的な検査手法を事例に適用しながら，手法の正当性と有効性に関して評価する.

5.2 STEP1の検証

STEP1では状態の制約を用いて再開可能状態の絞り込みを行なう. 新たな変数が作成され，新システム中にその変数の制約がある場合は，その変数の初期化した値を旧システムの制約中に組み込む. この手法の正当性と有効性を以下の項で確認する.

5.2.1 STEP1の正当性検証

この検査手法が正しいかどうかを判定するために手法が誤っていると仮定し，その時に考えられる不正な判定パターンを以下に記した.

pattern1. 判定をパスしたが更新後に不正な値の操作を行なう.

pattern2. 不正な値の操作を行なわないはずの再開ポイントを判定によって切り捨てる.

pattern1の考察

この手法では，更新後に不正な値の操作を行なう可能性を持つ候補状態を全て切り捨てる. 従ってもしpattern1のような判定をするケースがある場合は，以下の2つのケースが考えられる.

1. 制約の記述に漏れがあるケース.
2. 制約記述が誤りがあるケース.

今現在制約の記述は人為的に行なっているため，ケース1の問題は避けられない. 状態の制約を機械的に記述する手法の提案が必要である.

ケース2は状態が制約記述の条件を満たしていないが，そのまま記述されているという人為的なミスによるものである. このケースに関してはLTL式による検査で制約が正しいかどうかを検査することが出来る. 以下にはその検査に用いるLTL式を示す.

- □ (状態ラベル → 制約)

状態の制約とは，状態内に滞在している時に成り立つ条件の集合である．Promela 文で記述される状態ラベルを通る時は状態内に滞在していることを示す．従ってラベルを通る時に状態の制約が必ず成り立っていなければならない．上記の式は状態ラベルを通る時に状態の制約が常に成り立つかどうかを判定する．この検査でエラーが検出される場合は，制約文の成り立たないケースが存在していることを示しているので，その制約を除外する．この方法によりケース 2 を回避することが出来る．

pattern2 の考察

pattern2 は判定基準が厳しすぎるために起こる不正な判定である．この判定によって後々に不具合が発生させることはない．しかし更新可能な状態を見落としてしまうため，この手法の性能を低下させる可能性を秘めている．今現在は最も厳しい判定基準で判定を行なっているので，再開可能状態を見逃しているケースは大いにありうる．

事例においてもそのようなケースが発見された．事例において，STEP1 の判定をパスできる状態の組み合わせは以下の通りである．

- 電子レンジシステム
  - old.IDLE → new.IDLE
  - old.OPEN → new.OPEN
- マグネトロン
  - old.IDLE → new.IDLE
  - old.RUNNING → new.RUNNING.600W

ここで old.STATE → new.STATE とは，旧システムの状態から新システムの状態に向けて update 遷移を付加できることを示す． この事例で注目したいのが old.HEAT → new.RANGEHEAT の遷移が判定から漏れていることである．この 2 つの状態はどちらもレンジを稼動させている状態で，一見すると安全に再開できそうな状態の組である．判定から漏れた理由は new.RENGEHEAT 状態の制約 [[light == true]] を旧システムが満たさなかったためであった．light は更新後に新たに登場した変数で，初期化は更新時にライトが点灯していないという理由で light = false と代入された．しかし更新時にライトを点灯させ，light = true と代入し，ライトオブジェクトを ON 状態から開始させるのであれば，これ以降の段階的な検査を全てパスして，再開可能と判定できるのである．

現段階ではこのようなケースを容認するルールが制定されていないため，不正であると判定している．これは STEP1 の手法にまだ改善の余地があることを示している．判定基準を緩和することで，より有効な手法になるであろう．

5.2.2 STEP1 の有効性検証

この項では STEP1 の検査を使用しない場合とする場合を，事例を基に考察することによって手法の有効性を主張する.

STEP1 の検査を使用しない場合

STEP1 の検査をしない場合，更新前の各状態に対し，対応するオブジェクト内の全ての状態が候補状態となる. 事例では更新によって変化したオブジェクトとして，電子レンジシステムとマグネトロンオブジェクトがあげられる. 状態数はそれぞれ $3 \rightarrow 4$，$2 \rightarrow 3$ に変化した. 電子レンジシステムの各状態は４つの候補状態を持っていて，マグネトロンにおいては各状態が３つの候補状態を持っている.

STEP2 の検査は各オブジェクトがどの状態に滞在して時にデッドロックが起きないかを判定する. それを明確に調べるためには各オブジェクトに１本ずつの update 遷移を付加し，それらの組み合わせを網羅的に検査する必要がある. 従ってこのケースにおいて以後の検査を $3 \times 4 \times 2 \times 3$ で計 72 回行なわなければならない.

STEP1 の検査を使用する場合

STEP1 の検査を行なう場合，電子レンジシステムにおいての候補状態は IDLE 状態が１つ，OPEN 状態が 1 つの計２つ. またマグネトロンの候補状態は，IDLE 状態 1 つ，RUNNING 状態１つの計２つ. それらの全ての組み合わせをとっても以後の検査は 2×2 の計 4 回のみで済む.

考察

比較的簡単なこの事例においても，STEP1 の検査を使用するかどうかで以後の検査回数が 18 倍変わる. これがより複雑で状態数やオブジェクト数の多いシステムであった場合，この手法を使用しなければ検査回数が爆発的に増え，莫大なコストがかかる恐れがある. 前項で述べたように，絞込みを行なうことによって再開可能な状態を見落とす可能性はあるが，コスト面の利点を考慮に入れると明らかに有効であると言える.

5.3 STEP2 の検証

STEP2 では SPIN の到達性解析により，更新においてデッドロックが発生しないかどうかを検査する. この手法が誤っていると仮定した時に考えられる不正な判定を以下に記した.

- 判定をパスしたがデッドロックが起こる.

- デッドロックが検出されたが実際にはデッドロックが起きない.

5.3.1 STEP2 の正当性検証

SPIN は状態空間を網羅的に探索してシステムの性質を検査するツールである. 従って SPIN がデッドロックを検出できないということはありえない. もし考えられるとしたらシステムのモデル化に誤りがあるケースのみである. モデル化誤りの問題はこの手法を検証する上での問題にはならない. 上記のケースは両方ともモデル化に誤りがあるケースである. すなわちこの手法の不当性を示すものとはなりえないので, この手法は正当であると言える.

5.3.2 STEP2 の有効性検証

事例において, 更新によって変更されたオブジェクトは電子レンジシステムとマグネトロンの 2 つであった. これらのオブジェクト中には, STEP1 の検査をパスした再開可能状態の候補がそれぞれ 2 つずつ存在する. その詳細については前節で述べた通りである. このステップに入る前に新旧システムの状態遷移図を結合し, 旧システムの状態から候補の状態に向けて update 遷移を各オブジェクトに 1 本ずつ付加する. そして SPIN の到達性解析を用いてデッドロックが起きないかどうかを判定する. 図 5.4 は電子レンジシステムとマグネトロンの新旧状態遷移図を結合して, update 遷移を 1 本ずつ付加した例である. 図中の状態名のダッシュ記号 (’) は新システムの状態を表しており, 太い二重線の矢印は update 遷移を示している.

STEP2 の検査は各オブジェクトの update 遷移の組み合わせで検査を行なうため, ここでは 4 パターンの検査を行なった. そのうち STEP2 の検査をパスしたのは以下の 2 パターンであった.

- 電子レンジシステム　old.IDLE → new.IDLE
  マグネトロン　old.IDLE → new.IDLE
- 電子レンジシステム　old.OPEN → new.OPEN
  マグネトロン　old.IDLE → new.IDLE

それ以外のパターンはそれぞれのオブジェクトの更新ポイントがかみ合わないため, update 遷移によって新システムに遷移する前にデッドロックが検出された.

またこの手法では update 遷移によって全てのオブジェクトが新システムへの移行に成功したが, システム内のメッセージ通信がかみ合わずにデッドロックが発生するケースも検出できる. まとめるとこの検査では以下の 2 種類の不具合を検出できる.

- 更新ポイントがかみ合わないため, 新システムに移行することができず, システムがダウンする不具合.

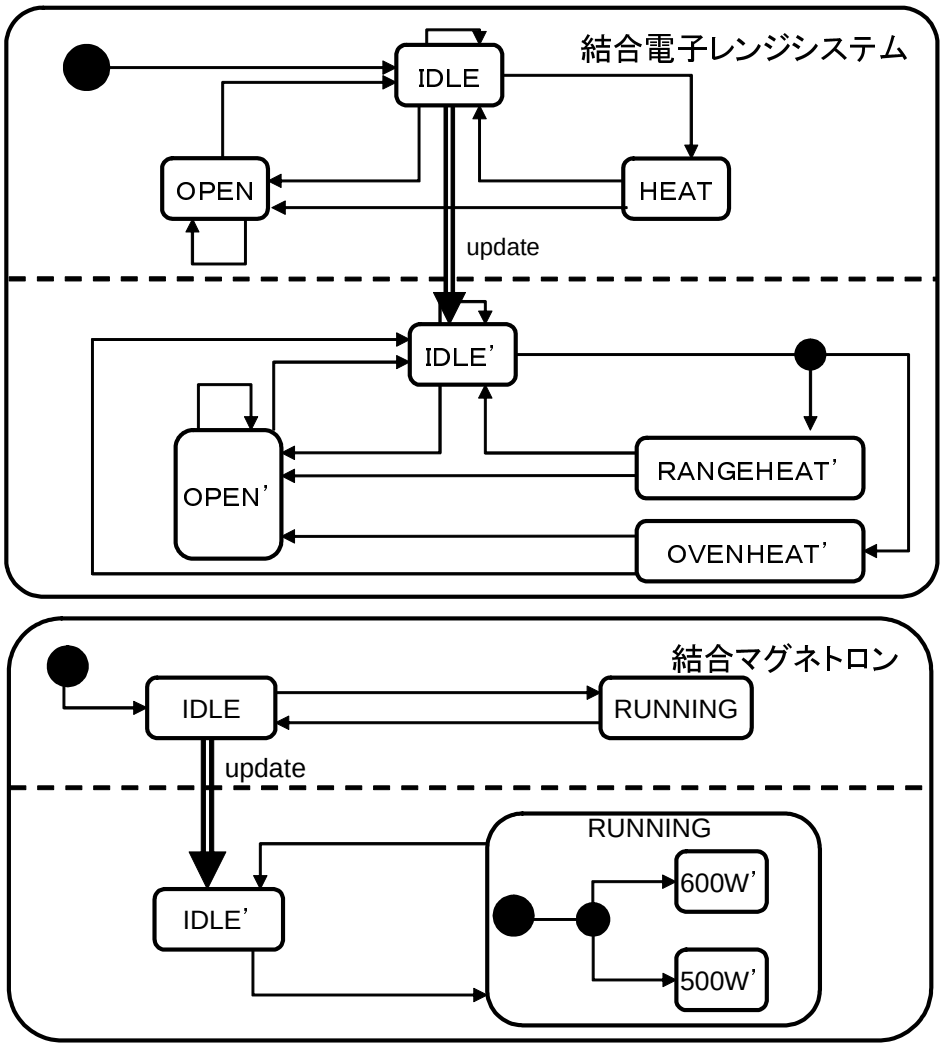
結合電子レンジシステム
IDLE
OPEN
HEAT
update
IDLE'
OPEN'
RANGEHEAT'
OVENHEAT'
結合マグネトロン
IDLE
RUNNING
update
RUNNING
IDLE'
600W'
500W'

図 5.4: 結合状態遷移図

- 更新後にメッセージ通信がかみ合わず，システムがダウンする不具合.

しかしながらこの手法自体が適用できないケースが 1 つだけ存在する. それはシステム内にオブジェクトが 1 つしかなく，オブジェクト間のメッセージ通信自体が行なわれないケースである. この場合は更新においてデッドロックが起こるはずもなく，検査は常に正しいと判定されるので，意味のない検査となる.

5.4 STEP3 の検証

STEP3 では新システムが旧システムの仕様を満足し，かつ更新においてサービスの欠落や重複提供が発生しないかどうかを検査する. 検査は LTL 式を用いて行なわれる. この節では検査に使用する 4 つの LTL 式が誤っていると仮定して，その時に起こりうる不正な判定を調べる. ここでは LTL 式を 1 つずつ検証することで，本手法の正当性を評価する. そしてこれらの式から得られる手法の有効性を示し，最後にこれまで紹介してきた事例の検査結果を示す.

5.4.1 仕様の充足性検査の正当性検証

第 1 式

1. □ ((サービス提供開始条件 && updateComp == true) → (◇提供されるサービス群))

この式は，新システムが旧システムの仕様を補完しているかどうかを調べる式である. エラーが出力されなければ，新システムは旧仕様を満足していると判定する.

以下には不正な判定を起こしうるケースを列挙する.

1. 条件部が常に成立しない.

   第 1 式の左側の条件式が常に成立しない場合，この式は正しいと判定される. それらのケースを以下に示す.

   (a) updateComp が常に偽である.

   (b) 仕様の条件部が更新後に常に偽である.

2. 不必要なサービスまで提供する.

1(a) のケースは update 遷移が正常に行なわれていないことを示す. しかし STEP2 の検査をパスしていることが前提なので，このケースが起こることはない.

1(b) のケースは更新後に旧仕様の条件を常に満たさないケースである. このケースはこの式では判定することができない. しかしこの式を補うのが第 2 式であり，第 2 式を適用することで問題は解決することができる.

2 のケースは不必要なサービスが提供されているが，正しいと判定するケースである．この式は新たに提供するサービスが付け加わることは許可しているので，この式だけで不必要なサービスが提供されたかどうかを判定することは不可能である．もし不必要かどうかを正確に調べたい場合には，次の LTL 式の検査を行なうべきであろう．

- □ ((仕様記述の条件部 && updateComp == true) →
  ◇ sservice[i] == 1 || ◇ service[j] == 1 || ...))

ここで service[n] は提供されるべきではないサービスのカウンタを示す．SPIN がエラーを出力しない場合には，不必要なサービスが提供されたことを示す．しかし何を不必要なサービスと定義づけるかは別問題である．従って本研究ではこの検査は行なわない．本研究では既存のサービスに新たなサービスが追加されるものについては全て正しいと判定する．その考えの下ではこの式は不正な判定をしないので，正当であると言える．

第 2 式

2. □ ((サービス提供開始条件 && updateComp == true) → false)

この式は第 1 式において，検査できない仕様の条件部が常に偽であるケースの検出を行なう．エラーが検出される場合は，条件部を満たす箇所が新システムに存在することを示すので正しいと判定する．エラーが検出されない場合は，条件部が常に偽であることを示しているので，新システムが旧仕様を満たしていないとして判断して候補状態から除去する．この式が不正な判定を行なうケースを考察したが，第 1 式であげたケース 1(a) のみであり，そのケースは STEP2 をパスしている段階で起こることはない．従ってこの式が不正な判定を行なうケースはないので，この式は正当であると言える．

5.4.2 仕様の充足性検査の有効性検証

上記の正当性の検証から第 1 式と第 2 式を併用すれば，新システムが旧システムの仕様を満足するか否かを判定することが可能である．従ってこの検査を行なうことで，ユーザが求める旧システムの動作が更新後にも補完されていることを証明することができる．

5.4.3 サービスの過不足検査の正当性検証

第 3 式

3. □ ((サービス提供開始条件) → (◇提供するサービス群))

この式は旧システムの仕様の LTL 式そのものである．ただし違うのは update 遷移が付加されているという点である．仮定から旧システムは単体では正しく動作するものであり，仕様は満たされている．また第１式をすでに満たしていることから，更新後にもその仕様が存在することがわかっている．従って第 3 式によって検査できるのは，更新時においてサービスの欠落が起きているか否かである．システムがエラーを出力する場合は更新時にサービスが欠落したことを示す．

この式で考えられる不正な判定は，仕様を常に満たさないケースであるが，上で述べたように新旧システムはすでにその仕様を満足していることから，このケースは起こりえない．それ以外に考えられる不正な判定を行なうケースが存在しなかったため，この式は正当であると言える．

第 4 式

4. □ (サービス 1 <= 1 && サービス 2 <= 1 && ...)

第４式は更新時にサービスが重複提供されていないかどうかを判定する式である．サービスのカウンタは，更新前と更新後は常に 0 か 1 の値しかとりえない．しかし更新時において図 3.8 のように過剰提供されるケースが存在する．第 4 式はこのケースを検出することができる．エラーが検出される場合は，重複提供が行なわれていることを示す．

この式の不正な判定を行なうケースを考察したが，そのようなケースは発見されなかった. よってこの式は正当であると言える．

5.4.4 サービスの過不足検査の有効性検証

これらの判定を行なうことで，ユーザの求めているサービスが欠落しないこと，また重複して提供されないことを証明し，更新時の安全性を高めることができる．

5.5 事例の検証結果

事例において STEP2 までの検査をパスすることができたのは，5.3.2 で示した 2 パターンである．この 2 つに STEP3 の検査をかけたところ，2 つのパターンは両方とも検査をパスすることができた. 従ってこの事例において動的な更新を安全に行なうこと判定できたのは，以下に示す 2 パターンである．

- 電子レンジシステムオブジェクトが IDLE 状態に滞在していて，かつマグネトロンオブジェクトが IDLE 状態に滞在している時.
- 電子レンジシステムオブジェクトが OPEN 状態に滞在していて，かつマグネトロンオブジェクトが IDLE 状態に滞在する時.

5.6 まとめ

最後に本章で行なった評価をまとめる．各ステップごとに分けて利点と欠点を述べた．

- STEP1:状態の制約による候補の絞り込み．

  この検査を行なうことによって，不正な値をとる状態からの再開を予防することができる．しかし状態の制約自体に漏れがある場合には正しい検査ができないため，不正な値をとる状態からの再開を許す可能性がある．現在，制約記述は人為的に行なっているため，制約記述の漏れは避けられない．機械的に制約を生成する手法の提案が必要である．

  また判定基準が高すぎることで，再開可能であるはずの状態を切り捨てる可能性を秘めている．しかしその一方で候補状態を大量に切り捨てるため，この後の段階の検査回数を大幅に削減できるというコスト面での利点が得られる．今後はこの利点を削り過ぎない程度で判定基準を緩和するルールの制定が必要である．

- STEP2:システムのデッドロック検査．

  この検査を行なうことで，更新後にデッドロックを起こす状態からの再開を回避することができる．検出されるデッドロックは以下の２種類である．

  – 各オブジェクトの再開可能な候補ポイントがかみ合わず，更新自体ができない場合に発生するデッドロック．
  – 更新は完了したがその後のオブジェクト間のメッセージ通信が不能となったために起こるデッドロック．

  しかしシステム内にオブジェクトが１つしかない場合はこの手法を適用することができない．オブジェクトが１つしかないため，更新ポイントがかみ合わなくなることも，メッセージ通信が不能になることも起こりえないためである．この場合，この手法は無意味な検査となる．

- STEP3：LTL 式による仕様検査．

  この検査では新システムが旧システムの仕様を満足し，更新時にサービスの欠落や重複提供を起こさないことが保証される．これによりユーザに不利益が生じる状態からの再開を回避することができる．

  問題点としてはサービスの欠落や重複提供に関しては検査することはできるが，不必要なサービスが提供されるケースは判定できない．今後，不必要なサービスを見分ける手法の提案が必要である．

第6章　おわりに

最後に本研究をまとめ，今後の展望について述べる．

6.1　研究総括

ソフトウェアの動的な更新を実現するためには，更新時にシステムが不具合を起こさずに再開可能である安全なポイントを発見する手法の提案と，そのポイントから現実的にシステムを移行する手法の提案が必要である．

本研究では前者について，状態遷移図を用いたアプローチで手法の提案を行なった．上記の不具合としてシステムの停止，または暴走，旧仕様の満足，サービスの過不足の発生の 4 つに焦点をあて，段階的に各々の不具合を検査する手法を提案した．ステップ 1 では不正な値の操作について検査し，ステップ 2 ではシステムの停止と暴走について検査する．そしてステップ 3 では旧仕様の満足，サービスの過不足の発生について検査する．この段階的な検査を全てパスする状態では，動的に再開可能であると判定する．検査にはモデル検査ツール SPIN を使用し，事例として電子レンジシステムを用いて研究を進めた．この手法の利点は以下の通りである．

- 検査に多大なコストがかからない．
- 更新によって矛盾が起こる状態からの再開を大幅に軽減することができる．
- 更新においてシステムがダウンしないことを検査することができる．
- ユーザの求める仕様を満足しているかどうかを検査することができる．
- ユーザに不利益が生じないことを証明できる．

また反対に欠点は以下の通りである．

- 旧システムで存在していた変数，サービス，システム外部から受理するイベントが失われる更新においてはこの手法を適用することができない．
- 判定基準が厳しすぎるため再開可能なポイントが発見できないことがある．
- システム内の全てのオブジェクトに少なくとも 1 つの再開可能ポイントが存在していなければならない．

欠点のうち，後半の 2 つは判定基準の修正によって改善できるものである．現在は最も厳しい判定基準で検査を行なっている．従って今後この基準を緩和するルールを制定することで，この手法はより有効なものへと変化するであろう．

6.2 今後の展望

先の節で述べたとおり，今後は判定基準を緩和する必要がある．この手法の最も大きなメリットは検査にコストがかからないことであるので，緩和のルールを制定する際には検査回数を増大させすぎないことに注意すべきである．

また本研究ではシステムが完全に状態遷移図通りに実装されていると仮定を置いた．しかし，現在一般的なシステムは完全に状態遷移図通りに実装されているわけではない．この手法を適用するには，状態遷移図通りにシステムを構築する手法の提案が必要である．更に，動作中のシステムから実行状態を取り出して新システムに与え，正しい箇所から再開させる具体的な方法の提案を行なう必要がある．

謝辞

本研究を進める上で終始御指導を賜り，精神面でも大きく支えて頂いた片山卓也教授に心から感謝致します. 本研究の根本を築いて下さった岡崎光隆氏に深く感謝の意を示します.

また，適切な指摘や助言をして頂いた青木利晃助手，林信宏氏に心より感謝致します.

そして，快適な学生生活の場を提供して頂いた，友人，知人，片山研究室，岸研究室，ならびに Defago 研究室の皆様に厚く御礼申し上げます.

最後に，ここまで学業に携わることを許して頂いた両親に深く感謝の意を表します.

皆様，本当にありがとうございました.

参考文献

[1] G.J.Holzmann, The SPIN MODEL CHECKER, Addison-Wesley, pp596, 2003.

[2] J. ランボー, M. ブラハ, W. プレメラニ, F. エディ, W. ローレンセン=著, 羽生田栄一=監訳, オブジェクト指向方法論 OMT, Prentice Hall, 凸版印刷株式会社, pp544, 1992.

[3] S. シュレィアー, S.J. メラー=著, 本位田真一, 伊藤潔=監訳, 続・オブジェクト指向システム分析, 啓学出版株式会社, pp265, 1992.

[4] http://spinroot.com/spin/Man/index.html

[5] 中島震 書評;The SPIN Model Checker, コンピュータソフトウェア, Vol21, No.2.

[6] http://research.nii.ac.jp/ hidaka/spin_chapter3

[7] 橋本政朋, ”ユビキタス環境を支えるプログラミング言語システム ”. 先進情報システムとその構成に向けて:「機能と構成」領域. 東京,2004-01, 科学技術振興機構, 東京,2004,p. 43-51.

[8] http://www.sw.nec.co.jp/products/itpoint/05/

付録

ここには本研究で使用した電子レンジの事例のソースコードを記載した.

```
/*****************************************************************
事例　電子レンジ
```

更新前のシステム概要

電子レンジには TIME ボタン，START ボタン，扉，ディスプレイ，マグネトロン，テーブルが備わっている．ユーザの行なえる動作は『TIME ボタンを押す』,『START ボタンを押す』,『扉を開ける』,『扉を閉める』の４つである．旧システムの仕様は次の通りである。

・電子レンジ停止中に TIME ボタンを押すと，温め時間が 10 秒増え，時間が表示される。
・電子レンジ停止中に，扉が閉まっており，温め時間がセットされている時に START ボタンを押すと，マイクロ波が照射され，テーブルが回転する．
・電子レンジ稼動中に温め時間が０になると，マイクロ波の照射が停止し，テーブルが止まり，ディスプレイが消える．
・電子レンジ稼動中に扉を開けると，マイクロ波の照射が停止し，テーブルが止まり，ディスプレイが消える．
・電子レンジ稼動中は温め時間のカウントダウンを行い、その都度時間を表示する．

更新内容

上記の旧システムにモード切替ボタン，電力切替ボタン，ライトを付加する．
更新によってオーブン機能が使用が可能となる．電子レンジ停止時にモード切替ボタンを押すことで電子レンジモードとオーブンモードを切替えることができる．
デフォルトは電子レンジモードとなっている．
また電子レンジ停止時に電力切替ボタンを押すと，
５００ｗと６００ｗを選択することができる．
デフォルトは６００ｗとなっている。
電子レンジ，オーブン稼動時はライトが点灯し内部が見渡せるようになる．

状態の制約

```
//////////////////////////////////////////////////
////  状態の制約の LTL 記述形式                  ////
////  [](s -> (c1 && c2...))                    ////
////  s          :   オブジェクト内の状態         ////
```

```
////  c1 , c2 ...：    状態に付加された制約          ////
//////////////////////////////////////////////////////
```

以下には各オブジェクト中の状態における制約を記述する.

```
//////////////////////////////////////////////////////////////////////
```

電子レンジシステム

```
～old version～

状態：  IDLE
制約：  {door == true && running == false}
LTL 式： [](s -> (c1 && c2))
#define s ElectricOven@IDLE
#define c1 door == true
#define c2 running == false


状態：  OPEN
制約：  {door == false && running == false}
LTL 式： [](s -> (c1 && c2))
#define s ElectricOven@OPEN
#define c1 door == false
#define c2 running == false


状態：  HEAT
制約：  {door == true && running == true}
LTL 式： [](s -> (c1 && c2 && c3))
#define s ElectricOven@HEAT
#define c1 door == true
#define c2 running == true
#define c3 time > 0


～new version～

状態：  IDLE_d
制約：  {door == true && running_d == false}
LTL 式： [](s -> (c1 && c2))
#define s ElectricOven@IDLE_d
#define c1 door == true
#define c2 running_d == false


状態：  OPEN_d:
制約：  {door == false && running_d == false}
```

```
LTL式： [](s -> (c1 && c2))
#define s ElectricOven@OPEN_d
#define c1 door == false
#define c2 running_d == false


状態： RANGEHEAT_d:
制約： {door == true && running_d == true &&
          emanation == true && oven == false && time > 0}
LTL式： [](s -> (c1 && c2 && c3 && c4 && c5))
#define s ElectricOven@RANGEHEAT_d
#define c1 door == true
#define c2 running_d == true
#define c3 emanation == true
#define c4 oven == false
#define c5 time > 0


状態： OVENHEAT_d:
制約： {door == true && running _d == true &&
          emanation == false && oven == false && time > 0}
LTL式： [](s -> (c1 && c2 && c3 && c4 && c5))
#define s ElectricOven@OVENHEAT_d
#define c1 door == true
#define c2 running_d == true
#define c3 emanation == false
#define c4 oven == true
#define c5 time > 0

////////////////////////////////////////////////////////////////////
マグネトロン

〜old version〜

状態： IDLE
制約： {emanation == false}
LTL式： [](s -> (c1))
#define s Magnetron@IDLE
#define c1 emanation == false


状態： RUNNING
制約： {emanation == true}
LTL式： [](s -> (c1))
#define s Magnetron@RUNNING
#define c1 emanation == true
```

～new version～

状態： IDLE_d:
制約： {emanation == false}
LTL 式： [](s -> (c1))
#define s Magnetron@IDLE_d
#define c1 emanation == false

状態： 600W_d
制約： {emanation == true && watt == 600}
LTL 式： [](s -> (c1 && c2))
#define s Magnetron@RUNNING600_d
#define c1 emanation == true
#define c2 watt == 600

状態： 500W_d:
制約： {emanation == true && watt == 500}
LTL 式： [](s -> (c1 && c2))
#define s Magnetron@RUNNING500_d
#define c1 emanation == true
#define c2 watt == 500

//
テーブル　（更新において変化なし）

状態： IDLE
制約： {turn == false}
LTL 式： [](s -> (c1))
#define s TurnTable@IDLE
#define c1 turn == false

状態： TURNING
制約： {turn == true}
LTL 式： [](s -> (c1))
#define s TurnTable@TURNING
#define c1 turn == true

//
ディスプレイ　（更新において変化なし）

状態： OFF

制約： {dispShow == false}
LTL 式： [](s -> (c1))
#define s Display@OFF
#define c1 dispShow == false

状態： ON
制約： {dispShow == true}
LTL 式： [](s -> (c1))
#define s Display@ON
#define c1 dispShow == true

//
電力切替装置　（新規追加）

状態： W600
制約： {watt == 600}
LTL 式： [](s -> (c1))
#define s WattSwitchingUnit@W600
#define c1 watt == 600

状態： W500
制約： {watt == 500}
LTL 式： [](s -> (c1))
#define s WattSwitchingUnit@W500
#define c1 watt == 500

//
モード切替装置　（新規追加）

状態： RANGE
制約： {mode == true}
LTL 式： [](s -> (c1))
#define s ModeSwitchingUnit@RANGE
#define c1 mode == true

状態： OVEN
制約： {mode == false}
LTL 式： [](s -> (c1))
#define s ModeSwitchingUnit@OVEN
#define c1 mode == false

//
ライト　（新規追加）

状態： OFF
制約： {light == false}
LTL 式： [](s -> (c1))
#define s Light@OFF
#define c1 light == false

状態： ON
制約： {light == true}
LTL 式： [](s -> (c1))
#define s Light@ON
#define c1 light == true

//
オーブン　　（新規追加）

状態： IDLE
制約： {oven == false}
LTL 式： [](s -> (c1))
#define s Oven@IDLE
#define c1 oven == false

状態： RUNNING
制約： {oven == true}
LTL 式： [](s -> (c1))
#define s Oven@RUNNING
#define c1 oven == true

****
****
旧システム仕様記述

#define u updateComp == true

電子レンジ停止中に TIME ボタンを押すと、温め時間が１０秒増え、
時間が表示される。
[]((e1 && c1 && c2) -> (<>s1 && <>s2))
#define e1 TIMEBUTTON
#define c1 running == false
#define c2 time < 20
#define s1 service[INCREASETIME] == 1
#define s2 service[SHOWDISPLAY] == 1

電子レンジ停止中に、扉が閉まっており、温め時間がセットされている時、
START ボタンを押すとマイクロ波が照射され、テーブルが回転する。

```
[]((e1 && c1 && c2 && c3) -> (<>s1 && <>s2))
#define e1 STARTBUTTON
#define c1 door == true
#define c2 running == false
#define c3 time > 0
#define s1 service[STARTWAVE600] == 1
#define s2 service[STARTROLLING] == 1
```

電子レンジ稼動中に温め時間が 0 になると、
マイクロ波の照射を停止し、テーブルを止め、ディスプレイを消す。

```
[]((c1 && c2 && c3) -> (<>s1 && <>s2 && <>s3))
#define c1 running  == true
#define c2 door == true
#define c3 time == 0
#define s1 service[STOPWAVE] == 1
#define s2 service[STOPROLLING] == 1
#define s3 service[CLEARDISPLAY] == 1
```

電子レンジ稼動中にドアを開けると、マイクロ波の照射を停止し、
テーブルを止め、ディスプレイを消す。

```
[]((e1 && c1 && c2) -> (<>s1 && <>s2 && <>s3))
#define e1 OPENDOOR
#define c1 door == true
#define c2 running == true
#define s1 service[STOPWAVE] == 1
#define s2 service[STOPROLLING] == 1
#define s3 service[CLEARDISPLAY] == 1
```

電子レンジ稼動中は温め時間のカウントダウンを行い,
その都度時間が表示される.

```
[]((c1 && c2 && c3 && l1) -> (<>s1 && <>s2))
#define c1 door == true
#define c2 running == true
#define c3 time > 0
#define l1    ElectricOven@INHEAT || ElectricOven@INRANGEHEAT_d
#define u updateComp == true
#define s1 service[COUNTDOWN] == 1
#define s2 service[SHOWDISPLAY] == 1

**********************************************************************/
```

```
/*イベン感知*/
/*START ボタンを押す*/
#define STARTBUTTON (
    ElectricOven@STARTBUTTON1 ||
    ElectricOven@STARTBUTTON2 ||
    ElectricOven@STARTBUTTON3 ||
    ElectricOven@STARTBUTTON1_d ||
    ElectricOven@STARTBUTTON2_d ||
    ElectricOven@STARTBUTTON3_d ||
    ElectricOven@STARTBUTTON4_d)

/*TIME ボタンを押す*/
#define TIMEBUTTON (
    ElectricOven@TIMEBUTTON1 ||
    ElectricOven@TIMEBUTTON2 ||
    ElectricOven@TIMEBUTTON3 ||
    ElectricOven@TIMEBUTTON1_d ||
    ElectricOven@TIMEBUTTON2_d ||
    ElectricOven@TIMEBUTTON3_d ||
    ElectricOven@TIMEBUTTON4_d)

/*扉を開ける*/
#define OPENDOOR (
    ElectricOven@OPENDOOR1 ||
    ElectricOven@OPENDOOR2 ||
    ElectricOven@OPENDOOR1_d ||
    ElectricOven@OPENDOOR2_d ||
    ElectricOven@OPENDOOR3_d)

/*扉を閉める*/
#define CLOSEDOOR (
    ElectricOven@CLOSEDOOR1 ||
    ElectricOven@CLOSEDOOR1_d)

/*電力切替ボタンを押す*/
#define WATTBUTTON (
    WattSwitchingUnit@WATTBUTTON1_d ||
    WattSwitchingUnit@WATTBUTTON2_d)

/*モード切替ボタンを押す*/
#define R_OBUTTON (
    ModeSwitchingUnit@R_OBUTTON1_d ||
    ModeSwitchingUnit@R_OBUTTON2_d)
```

```
/*旧バージョンレンジ稼動中フラグ*/
#define running (emanation == true && turn == true
    && dispShow == true)

/*新バージョンレンジ稼動中フラグ*/
#define running_d ((emanation == true || oven == true)
    && turn == true && dispShow == true && light == true)

/*サービスの総数*/
#define SERVICENUM 17
/*サービスカウンタ配列*/
byte service[SERVICENUM];

/*サービスの定義*/
#define INCREASETIME 0   /*時間を１０秒増やす*/
#define SHOWDISPLAY 1    /*時間を表示する*/
#define CLEARDISPLAY 2   /*時間表示を消す*/
#define STARTWAVE600 3   /*600w マイクロ波放出開始する*/
#define STOPWAVE    4    /*マイクロ波放出停止する*/
#define STARTROLLING 5   /*テーブルを回転する*/
#define STOPROLLING 6    /*テーブルを停止する*/
#define COUNTDOWN 7      /*温め時間を１秒減らす*/
/*新バージョンから追加されたサービス*/
#define LIGHTON 8        /*ライトをつける*/
#define LIGHTOFF 9       /*ライトを消す*/
#define STARTWAVE500 10  /*500w マイクロ波放出開始する*/
#define SET600W 11       /*ワット数を６００にセットする*/
#define SET500W 12       /*ワット数を５００にセットする*/
#define MODEOVEN 13      /*稼動モードをオーブンにする*/
#define MODERANGE 14     /*稼動モードをレンジにする*/
#define STARTOVEN 15     /*オーブンを稼動する*/
#define STOPOVEN 16      /*オーブンをとめる*/

bool emanation;    /*マイクロ波放射状態　放射中：true　停止中：false*/
bool turn;         /*回転テーブルの状態　回転中：true　停止中：false*/
bool dispShow;     /*ディスプレイの表示状態　表示中：true　消灯中：false*/
int time;          /*温め時間*/
bool door;         /*ドアの状態　閉：true　開：false*/
bool light;        /*ライトの状態　点灯：true　消灯：false*/
int watt;          /*電力の値*/
bool oven;         /*オーブンの稼動状態　稼動中：true　停止中：false*/
bool mode;         /*モード　電子レンジ：true　オーブン：false*/

/*イベン受理許可フラグ*/
bool ok_eo;
bool ok_w;
```

```
bool ok_m;

/*LTL 式チェック用変数*/
bool test  = false;

/*更新完了フラグ*/
bool updateComp;
#define u updateComp = true

/*ユーザ～電子レンジシステム間通信路*/
mtype = {pushStartButton, openDoor, closeDoor, pushTimeButton};
chan u_e = [0] of {mtype};

/*電子レンジシステム～マイクロ波放射装置間通信路*/
mtype = {emitWave, stopWave};
chan e_m = [0] of {mtype};

/*電子レンジシステム～回転テーブル間通信路*/
mtype = {turnTable, stopTable};
chan e_t = [0] of {mtype};

/*電子レンジシステム～ディスプレイ間通信路*/
mtype = {showTime, clearDisplay};
chan e_d = [0] of {mtype};

/*電子レンジシステム～ライト間通信*/
mtype = {turnOn, turnOff};
chan e_l = [0] of {mtype};

/*ユーザ～電力切替え装置間通信*/
mtype = {pushWattButton};
chan u_w = [0] of {mtype};

/*電子レンジシステム～電力切替装置間通信*/
mtype = {default};
chan e_w = [0] of {mtype};

/*電子レンジシステム～オーブン間通信*/
mtype = {startOven, stopOven};
chan e_o = [0] of {mtype};

/*ユーザ～モード切替装置*/
mtype = {pushR_OButton};
chan u_mo = [0] of {mtype};

/*電子レンジシステム～モード切替装置間通信*/
```

```
mtype = {modeDefault};
chan e_mo = [0] of {mtype};

/*アップデート用通信路*/
mtype = {update};
chan u_m = [0] of {mtype};

/*************************************************************************
ユーザ
*************************************************************************/
proctype User(){
HEAD:
    /*イベント送信が可能であるかの判定*/
    ok_eo == true;

    {if
    /*start ボタンを押す*/
    ::u_e!pushStartButton;
    /*time ボタンを押す*/
    ::u_e!pushTimeButton;
    /*扉を開ける*/
    ::u_e!openDoor;
    /*扉を閉める*/
    ::u_e!closeDoor;
    /*更新作業*/
    ::u_e!update ->
        u_m!update;
        goto UPDATE;
    fi} unless {ok_eo == false};

    goto HEAD;

UPDATE:
    /*更新時の変数の初期化*/
    light = false;
    watt = 600;
    oven = false;
    mode = true;
    ok_w = false;
    ok_m = false;


    run Light();
    run WattSwitchingUnit();
    run ModeSwitchingUnit();
    run Oven();
```

```
    updateComp = true;
    goto HEAD_d;

HEAD_d:
    /*イベント送信可能であるかの判定*/
    (ok_eo == true && ok_w == true && ok_m == true);

    {if
    ::u_e!pushStartButton;
    ::u_e!pushTimeButton;
    ::u_e!openDoor;
    ::u_e!closeDoor;
    /*新規追加イベント*/
    /*ユーザはこの機能の使用方法を知らないためコメントアウト*/
    /*
    ::u_w!pushWattButton;
    ::u_mo!pushR_OButton;
    */
    fi} unless {ok_eo == false || ok_w == false || ok_m == false};

    goto HEAD_d;

}

/*****************************************************************************************
サービスカウンタクリアメソッド
*****************************************************************************************/
inline clearService(){
    int i;
    atomic{
        /*全てのサービスカウンタ変数の値の初期化*/
         i = 0;
        do
        ::(i == SERVICENUM) -> break;
        ::else ->
            service[i] = false;
            i = i + 1;
        od;
    }
}


/*****************************************************************
電子レンジシステム
*****************************************************************/
proctype ElectricOven(){
```

```
/*状態 IDLE*/
IDLE:
    ok_eo = true;
    if
    /*start ボタンが押される*/
    ::atomic{u_e?pushStartButton ->
        ok_eo = false};
STARTBUTTON1:
        clearService();
        if
        ::(time > 0) ->
            e_m!emitWave;
            e_t!turnTable;
            goto HEAT;
        ::else -> goto IDLE;
        fi;

    /*time ボタンが押される*/
    ::atomic{u_e?pushTimeButton ->
        ok_eo = false};
TIMEBUTTON1:
        clearService();
        if
        ::(time < 20) ->
            service[INCREASETIME] =
                service[INCREASETIME] + 1;
            time = time + 10;
            e_d!showTime;
            goto IDLE;
        ::else -> goto IDLE;
        fi;

    /*扉が開けられる*/
    ::atomic{u_e?openDoor ->
        ok_eo = false};
OPENDOOR1:
        door = false;
        goto OPEN;
/*
    ::u_e?update ->
        goto IDLE_d;
*/

    fi;
```

```
/*状態 OPEN*/
OPEN:
    ok_eo = true;
    if
    /*start ボタンが押される*/
    ::atomic{u_e?pushStartButton ->
        ok_eo = false};
STARTBUTTON2:
        goto OPEN;

    /*time ボタンが押される*/
    ::atomic{u_e?pushTimeButton ->
        ok_eo = false};
TIMEBUTTON2:
        clearService();
        if
        ::(time < 20) ->
            service[INCREASETIME] =
                service[INCREASETIME] + 1;
            time = time + 10;
            e_d!showTime;
            goto OPEN;
        ::else -> goto OPEN;
        fi;

    /*扉が開けられる*/
    ::atomic{u_e?closeDoor ->
        ok_eo = false};
CLOSEDOOR1:
        door = true;
        goto IDLE;

    ::u_e?update ->
        goto OPEN_d;


    fi;

/*状態 HEAT*/
/*状態 HEAT のイベント受理フェイズ*/
HEAT:
    ok_eo = true;
     if
    /*start ボタンが押される*/
    ::atomic{u_e?pushStartButton ->
        ok_eo = false};
```

```
STARTBUTTON3:
        goto INHEAT;

    /*time ボタンが押される*/
    ::atomic{u_e?pushTimeButton ->
        ok_eo = false};
TIMEBUTTON3:
        goto INHEAT;

    /*扉が開けられる*/
    ::atomic{u_e?openDoor ->
        ok_eo = false};
OPENDOOR2:
        clearService();
        door = false;
        time = 0;
        e_m!stopWave;
        e_t!stopTable;
        e_d!clearDisplay;
        goto OPEN;
    ::ok_eo = false ->
        goto INHEAT;
    fi;

/*HEAT 状態の活動フェイズ*/
INHEAT:
    clearService();
    service[COUNTDOWN] = service[COUNTDOWN] + 1;
    time = time - 5;
    e_d!showTime;

    if
    /*温め時間が 0 となる*/
    ::(time == 0) ->
        clearService();
        e_m!stopWave;
        e_t!stopTable;
        e_d!clearDisplay;
        goto IDLE;
    ::else -> skip;
    fi;
    goto HEAT;

/**************************更新後*********************************/
/*状態 IDLE_d*/
IDLE_d:
```

```
    ok_eo = true;
    if
    /*start ボタンが押される*/
    ::atomic{u_e?pushStartButton ->
        ok_eo = false};
STARTBUTTON1_d:
        clearService();
        if
        ::(time > 0) ->
            if
            ::(mode == true) ->
                e_m!emitWave;
                e_t!turnTable;
                e_l!turnOn;
                goto RANGEHEAT_d;
            ::(mode == false) ->
                e_o!startOven;
                e_t!turnTable;
                e_l!turnOn;
                goto OVENHEAT_d;
            fi;
        ::else -> goto IDLE_d;
        fi;

    /*time ボタンが押される*/
    ::atomic{u_e?pushTimeButton ->
        ok_eo = false};
TIMEBUTTON1_d:
        clearService();
        if
        ::(time < 20) ->
            time = time + 10;
            service[INCREASETIME] =
                service[INCREASETIME] + 1;
            e_d!showTime;
            goto IDLE_d;
        ::else ->
            goto IDLE_d;
        fi;

    /*扉が開けられる*/
    ::atomic{u_e?openDoor ->
        ok_eo = false};
OPENDOOR1_d:
        door = false;
        goto OPEN_d;
```

```
    fi;

/*状態 OPEN*/
OPEN_d:
    ok_eo = true;
    if
    /*start ボタンが押される*/
    ::atomic{u_e?pushStartButton ->
        ok_eo = false};
STARTBUTTON2_d:
        goto OPEN_d;

    /*time ボタンが押される*/
    ::atomic{u_e?pushTimeButton ->
        ok_eo = false};
TIMEBUTTON2_d:
        clearService();
        if
        ::(time < 20) ->
            service[INCREASETIME] =
                service[INCREASETIME] + 1;
            time = time + 10;
            e_d!showTime;
            goto OPEN_d;
        ::else -> goto OPEN_d;
        fi;

    /*ドアが閉められる*/
    ::atomic{u_e?closeDoor ->
        ok_eo = false};
CLOSEDOOR1_d:
        door = true;
        goto IDLE_d;
    fi;

/*状態 RANGEHEAT_d*/
/*状態 RANGEHEAT_d のイベント受理フェイズ*/
RANGEHEAT_d:
    ok_eo = true;
     if
    /*start ボタンが押される*/
    ::atomic{u_e?pushStartButton ->
        ok_eo = false};
STARTBUTTON3_d:
        goto INRANGEHEAT_d;
```

```
    /*time ボタンが押される*/
    ::atomic{u_e?pushTimeButton ->
        ok_eo = false};
TIMEBUTTON3_d:
        goto INRANGEHEAT_d;

    /*扉が開けられる*/
    ::atomic{u_e?openDoor ->
        ok_eo = false};
OPENDOOR2_d:
        clearService();
        door = false;
        time = 0;
        e_m!stopWave;
        e_t!stopTable;
        e_d!clearDisplay;
        e_l!turnOff;
        e_w!default;
        e_mo!modeDefault;
        goto OPEN_d;

    ::ok_eo = false ->
        goto INRANGEHEAT_d;
    fi;

/*RANGEHEAT の活動フェイズ*/
INRANGEHEAT_d:
    clearService();
    service[COUNTDOWN] = service[COUNTDOWN] + 1;
    time = time - 10;
    e_d!showTime;

    if
    /*温め時間が 0 となる*/
    ::(time == 0) ->
        clearService();
        e_m!stopWave;
        e_t!stopTable;
        e_d!clearDisplay;
        e_l!turnOff;
        e_w!default;
        e_mo!modeDefault;
        goto IDLE_d;
    ::else -> skip;
    fi;
    goto RANGEHEAT_d;
```

```
/*ver6 new*/
/*状態 OVENHEAT*/
OVENHEAT_d:
    ok_eo = true;

     if
    /*start ボタンが押される*/
    ::atomic{u_e?pushStartButton ->
        ok_eo = false};
STARTBUTTON4_d:
        goto INOVENHEAT_d;

    /*time ボタンが押される*/
    ::atomic{u_e?pushTimeButton ->
        ok_eo = false};
TIMEBUTTON4_d:
        goto INOVENHEAT_d;

    /*扉が開けられる*/
    ::atomic{u_e?openDoor ->
        ok_eo = false};
OPENDOOR3_d:
        clearService();
        door = false;
        time = 0;
        e_o!stopOven;
        e_t!stopTable;
        e_d!clearDisplay;
        e_l!turnOff;
        e_w!default;
        e_mo!modeDefault;
        goto OPEN_d;

    ::ok_eo = false ->
        goto INOVENHEAT_d;
    fi;

/*OVENHEAT の活動フェイズ*/
INOVENHEAT_d:
    clearService();
    service[COUNTDOWN] = service[COUNTDOWN] + 1;
    time = time - 10;
    e_d!showTime;

    if
```

```
    /*温め時間が0となる*/
    ::(time == 0) ->
        clearService();
        e_o!stopOven;
        e_t!stopTable;
        e_d!clearDisplay;
        e_l!turnOff;
        e_w!default;
        e_mo!modeDefault;
        goto IDLE_d;
    ::else -> skip;
    fi;
    goto OVENHEAT_d;
}


/*********************************************************************
マグネトロン
*********************************************************************/
proctype Magnetron(){

/*状態 IDLE*/
IDLE:
    if
    ::atomic{e_m?emitWave ->
        service[STARTWAVE600] = service[STARTWAVE600] + 1;
        printf("マイクロ波放出開始\n");
        emanation = true;
        goto RUNNING;}

    ::u_m?update ->
        goto IDLE_d;

    fi;

/*状態 RUNNING*/
RUNNING:
    if
    ::atomic{e_m?stopWave ->
        service[STOPWAVE] = service[STOPWAVE] + 1;
        printf("マイクロ波放出停止\n");
        emanation = false;
        goto IDLE;}
/*
    ::u_m?update ->
        goto RUNNING600_d;
```

```
*/

    fi;


/***************************更新後*****************************/
/*状態 IDLE_d*/
IDLE_d:
    if
    ::atomic{e_m?emitWave ->
        if
        ::(watt == 600) ->
            service[STARTWAVE600] =
                service[STARTWAVE600] + 1;
            printf("600w マイクロ波放出開始\n");
            emanation = true;
            goto RUNNING600_d;

        ::(watt == 500) ->
            service[STARTWAVE500] =
                service[STARTWAVE500] + 1;
            printf("500w マイクロ波放出開始\n");
            emanation = true;
            goto RUNNING500_d;
        fi;}
    fi;

/*状態 RUNNING600_d*/
RUNNING600_d:
    if
    ::atomic{e_m?stopWave ->
        service[STOPWAVE] = service[STOPWAVE] + 1;
        printf("マイクロ波放出停止\n");
        emanation = false;
        goto IDLE_d;}
    fi;

/*状態 RUNNING500_d*/
RUNNING500_d:
    if
    ::atomic{e_m?stopWave ->
        service[STOPWAVE] = service[STOPWAVE] + 1;
        printf("マイクロ波放出停止\n");
        emanation = false;
        goto IDLE_d;}
    fi;
```

```
}


/*********************************************************************
回転テーブル
*********************************************************************/
proctype TurnTable(){

/*状態 IDLE*/
IDLE:
    if
    ::atomic{e_t?turnTable ->
        service[STARTROLLING] = service[STARTROLLING] + 1;
        printf("テーブル回転\n");
        turn = true;
        goto TURNING;}
    fi;

/*状態 TURNING*/
TURNING:
    if
    ::atomic{e_t?stopTable ->
        service[STOPROLLING] = service[STOPROLLING] + 1;
        printf("テーブル停止\n");
        turn = false;
        goto IDLE;}
    fi;

}


/*********************************************************************
ディスプレイ
*********************************************************************/
proctype Display(){

/*状態 OFF*/
OFF:
    if
    ::atomic{e_d?showTime ->
        service[SHOWDISPLAY] = service[SHOWDISPLAY] + 1;
        dispShow = true;
        printf("時間  %d\n", time);
        goto ON;}
    fi;
```

```
/*状態 ON*/
ON:
    if
    ::atomic{e_d?showTime ->
        service[SHOWDISPLAY] = service[SHOWDISPLAY] + 1;
        printf("時間  %d\n", time);
        goto ON;}
    ::atomic{e_d?clearDisplay ->
        service[CLEARDISPLAY] = service[CLEARDISPLAY] + 1;
        dispShow = false;
        printf("表示を消す\n");
        goto OFF;}
    fi;
}


/**********************************************************************
ライト
**********************************************************************/
proctype Light(){

    /*更新時の動作*/
    if
    ::(light == true) ->
        printf("ライト点灯\n");
        service[LIGHTON] = service[LIGHTON] + 1;
        goto ON;

    ::else -> goto OFF;
    fi;

/*状態 OFF*/
OFF:
    if
    ::atomic{e_l?turnOn ->
        light = true;
        printf("ライト点灯\n");
        service[LIGHTON] = service[LIGHTON] + 1;
        goto ON;}
    fi;

/*状態 ON*/
ON:
    if
    ::atomic{e_l?turnOff ->
        light = false;
```

```
        printf("ライト消灯\n");
        service[LIGHTOFF] = service[LIGHTOFF] + 1;
        goto OFF;}
    fi;

}

/*************************************************************************
電力切替装置
*************************************************************************/
proctype WattSwitchingUnit(){

    /*更新時の動作*/
    if
    ::(watt == 600) -> goto W600;
    ::(watt == 500) -> goto W500;
    fi;

/*状態 W600*/
W600:
    ok_w = true;
    if
    /*ワット切替ボタンが押される*/
    ::atomic{u_w?pushWattButton ->
        ok_w = false};
WATTBUTTON1:    clearService();
        if
        ::(emanation == false && oven == false) ->
            service[SET500W] = service[SET500W] + 1;
            watt = 500;
            goto W500;
        ::else -> goto W600;
        fi;

    /*電力をデフォルトに設定しなおす*/
    ::atomic{e_w?default ->
        service[SET600W] = service[SET600W] + 1;
        goto W600;}
    fi;

/*状態 W500*/
W500:
    ok_w = true;
    if
    /*ワット切替ボタンが押される*/
    ::atomic{u_w?pushWattButton ->
```

```
        ok_w = false};
WATTBUTTON2:
        clearService();
        if
        ::(emanation == false && oven == false) ->
            service[SET600W] = service[SET600W] + 1;
            watt = 600;
            goto W600;
        ::else ->
            goto W500;
        fi;

    /*電力をデフォルトに設定しなおす*/
    ::atomic{e_w?default ->
        service[SET600W] = service[SET600W] + 1;
        watt = 600;
        goto W600};
    fi;
}

/*********************************************************************
モード切替装置
*********************************************************************/
proctype ModeSwitchingUnit(){

    /*更新時の動作*/
    if
    ::(mode == true) -> goto RANGE;
    ::else -> goto OVEN;
    fi;

/*状態 RANGE*/
RANGE:
    ok_m = true;
    if
    /*モード切替ボタンが押される*/
    ::atomic{u_mo?pushR_OButton ->
        ok_m = false};
R_OBUTTON1:
        clearService();
        if
        ::(emanation == false && oven == false) ->
            service[MODEOVEN] = service[MODEOVEN] + 1;
            mode = false;
            goto OVEN;
        ::else -> goto RANGE;
```

```
        fi;

    /*モードをデフォルトに設定しなおす*/
    ::atomic{e_mo?modeDefault ->
        service[MODERANGE] = service[MODERANGE] + 1;
        goto RANGE;}
    fi;

/*状態 OVEN*/
OVEN:
    ok_m = true;
    if
    /*モード切替ボタンが押される*/
    ::atomic{u_mo?pushR_OButton ->
        ok_m = false};
R_OBUTTON2:
        if
        ::(emanation == false && oven == false) ->
            clearService();
            service[MODERANGE] = service[MODERANGE] + 1;
            mode = true;
            goto RANGE;
        ::else -> goto OVEN;
        fi;

    /*モードをデフォルトに設定しなおす*/
    ::atomic{e_mo?modeDefault ->
        mode = true;
        service[MODERANGE] = service[MODERANGE] + 1;
        goto RANGE;}
    fi;

}

/*********************************************************************
オーブン
*********************************************************************/
proctype Oven(){

/*状態 IDLE*/
IDLE:
    if
    ::atomic{e_o?startOven ->
        service[STARTOVEN] = service[STARTOVEN] + 1;
        printf("オーブン作動\n");
        oven = true;
```

```
        goto RUNNING;}
    fi;

/*状態 RUNNING*/
RUNNING:
    if
    ::atomic{e_o?stopOven ->
        service[STOPOVEN] = service[STOPOVEN] + 1;
        printf("オーブン停止\n");
        oven = false;
        goto IDLE;}
    fi;
}

/*************************************************************************
開始
*************************************************************************/

init {
    atomic{
        /*変数の初期化*/
        door = true;
        emanation = false;
        turn = false;
        dispShow = false;
        ok_eo = false;
        updateComp = false;

        run User();
        run ElectricOven();
        run Magnetron();
        run TurnTable();
        run Display();
    }
}
```