

NuSMVを用いた  
CTL\*のモデル検査器の実装

東京工業大学  
理学部  
情報科学科

藤原一貴  
(0521236)

平成20年度卒業論文

指導教官 佐々 政孝 教授

2009年2月5日

# 目次

第1章	はじめに	4
1.1	背景	4
1.2	研究動機	4
1.3	研究概要	4
第2章	準備	5
2.1	時相論理	5
2.1.1	クリプキ構造 – Kripke structure	5
2.1.2	線形時相論理 – LTL	5
2.1.3	計算木論理 – CTL	7
2.1.4	CTL*	9
2.2	モデル検査	11
2.3	NuSMV	12
2.3.1	モデル検査の概要	12
2.3.2	smv ファイル	12
2.3.3	NuSMV での LTL 検査	14
2.4	Buchi オートマトン	15
2.4.1	定義	15
2.4.2	拡張 Buchi オートマトン	16
2.4.3	Buchi 受理言語の和集合, 積集合, 補集合	17
第3章	CTL*モデル検査器の設計と実装	19
3.1	アルゴリズム	19
3.1.1	LTL モデル検査のアルゴリズム	19
3.1.2	CTL モデル検査のアルゴリズム	20
3.1.3	CTL*モデル検査のアルゴリズム	23
3.2	実装	24
3.2.1	システム	24
3.3	CTL*モデル検査の例	25
3.3.1	例. 電子レンジ	25
第4章	関連研究	31
4.1	モデル検査器	31
4.1.1	SPIN	31
4.1.2	UPPAAL	31
4.1.3	LTSA	31

4.2	応用研究 . . . . .	32
4.2.1	Lacey らの研究 . . . . .	32
4.2.2	佐原の研究 . . . . .	32
第5章	まとめ	33

# 目 次

2.1	LTL 式の例 . . . . .	7
2.2	CTL 式の例 . . . . .	9
2.3	NuSMV の操作の流れ . . . . .	12
2.4	例の状態遷移図 . . . . .	13
3.1	LTL モデル検査アルゴリズムの概略図 . . . . .	19
3.2	本システムの概略図 . . . . .	24
3.3	電子レンジの状態遷移図 . . . . .	26
3.4	$x_0$ の更新をしたクリプキ構造 . . . . .	30

# 第1章 はじめに

## 1.1 背景

昨今，ハードウェアやソフトウェアの設計から導出されたモデルの検証を，モデル検査を用いて行う手法が広く知られるようになった．その際，一般的なモデル検査において，モデルの検査仕様は時相論理を用いて表すことが多く，そこで使用される時相論理の種類は，主に LTL[12] や CTL[3] といわれるものである．それに伴い，一般的に広まっているモデル検査器（SPIN[13][6]，NuSMV[11]）についても，検査可能な時相論理式は，主に LTL と CTL である．

## 1.2 研究動機

現在，このモデル検査の適用はハードウェアの設計において多く見られるが，ソフトウェアの設計におけるモデル検査の適用が，今後増えていくであろう．その際，モデルの検査仕様をこの LTL や CTL よりも記述力の高い言語で表し，その仕様を検査できるモデル検査器の必要性がでてくる．現に，コンパイラの最適化器の検証をモデル検査によって行う研究 [17] において，最適化器によっては仕様を LTL や CTL では表しきれないものも出てきている．

## 1.3 研究概要

本研究では，LTL と CTL の検査ができるモデル検査器 NuSMV を使用して，LTL と CTL より真に広いクラスを扱う CTL\*[10] の検査が可能なモデル検査器のシステムの構築と実装を行った．様々な最適化を施された NuSMV の複雑な中身を壊さぬよう安全に実装を行うという考えから，NuSMV を外側から使用するシステムで実装を行っている．入力には NuSMV の入力ファイルを用いるため，NuSMV を使用したことがある人にとって，使いやすいものになっている。

## 第2章 準備

本論文の説明にあたって時相論理とモデル検査の知識が必要となる。本章ではこれに関する説明を行う。

### 2.1 時相論理

時相論理は様相論理の一種であり、状態遷移システム上で成り立つ性質を記述するのに適した論理である。時相論理には様々な種類が存在するが、基本的に Kripke 構造上において意味を定義される。この節では本研究に関連するいくつかの時相論理を説明する。

#### 2.1.1 クリプキ構造 — Kripke structure

非決定性有限オートマトンの一種で、主にモデル検査に使用される。クリプキ構造は3つ組  $M = (S, R, L)$  で定義される。

- $S$ : 非空な状態の有限集合
- $R \subseteq S \times S$ : 状態集合上の2項関係
- $L: S \rightarrow 2^{AP}$ :  $AP$  を原始命題全体の集合としたとき、 $L$  は各状態で真になる命題変数の集合を与える<sup>1</sup>。これをラベリング関数という。

これは、以下に示す時相論理の意味論を表すために必要となる。

#### 2.1.2 線形時相論理 — LTL

線形時間時相論理 (Linear-time Temporal Logic, LTL) [12] とは、あるクリプキ構造の単一の経路に関する性質を記述する論理である。

クリプキ構造の経路を時系列だと考えれば、単一の経路は (分岐がないという意味で) 線形時間であり、線形時間を扱う論理を特に線形時間論理という。LTL は線形時間論理である。

---

<sup>1</sup>つまり  $\alpha \in L(s)$  ならば、原始命題  $\alpha$  は状態  $s$  で成り立つ。

## LTL の構文

LTL の構文は表 2.1 の通りである . ただし , *proposition* は原始命題を表す .

$$\begin{array}{l}
 \textit{LTL-formula} ::= \textit{proposition} \\
 | \neg \textit{LTL-formula} \\
 | \textit{LTL-formula} \wedge \textit{LTL-formula} \\
 | \textit{LTL-formula} \vee \textit{LTL-formula} \\
 | \textit{LTL-formula} \rightarrow \textit{LTL-formula} \\
 | X \textit{LTL-formula} \\
 | F \textit{LTL-formula} \\
 | G \textit{LTL-formula} \\
 | \textit{LTL-formula} U \textit{LTL-formula}
 \end{array}$$

表 2.1: LTL の構文規則

ここで ,  $X, F, G, U$  は時相演算子と呼ばれるものであり , それぞれ , *neXt* , *Future* , *Globally* , *Until* という英単語に由来している .

## LTL の意味論

Kripke 構造  $M = (S, R, L)$  の経路  $p = s_0 \rightarrow s_1 \rightarrow \dots$  で LTL 式  $\phi$  が成り立つことを ,  $M, p \models \phi$  , または  $M$  を省略して  $p \models \phi$  と表す . 時相演算子の直観的な意味は次の通りである .

$$\begin{array}{ll}
 p \models X\phi & : \text{経路 } p \text{ の先頭の次の状態で } \phi \text{ が成立する .} \\
 p \models F\phi & : \text{経路 } p \text{ 上でいつか } \phi \text{ が成立する .} \\
 p \models G\phi & : \text{経路 } p \text{ 上の全ての状態で } \phi \text{ が成立する .} \\
 p \models \phi_1 U \phi_2 & : \text{経路 } p \text{ 上で , } \phi_2 \text{ が成立するまで } \phi_1 \text{ が成立し続ける .}
 \end{array}$$

LTL の正確な意味の定義は , 表 2.2 の通りである . ただし ,  $p_i = s_i \rightarrow s_{i+1} \rightarrow \dots$  とする .

$$\begin{array}{ll}
 p \models \textit{proposition} & \text{iff } \textit{proposition} \in L(s_0) \\
 p \models \neg\phi & \text{iff not } p \models \phi \\
 p \models \phi_1 \wedge \phi_2 & \text{iff } p \models \phi_1 \text{ and } p \models \phi_2 \\
 p \models \phi_1 \vee \phi_2 & \text{iff } p \models \phi_1 \text{ or } p \models \phi_2 \\
 p \models X\phi & \text{iff } p_1 \models \phi \\
 p \models F\phi & \text{iff there exists a } i \geq 0 \text{ such that } p_i \models \phi \\
 p \models G\phi & \text{iff for any } i \geq 0, p_i \models \phi \\
 p \models \phi_1 U \phi_2 & \text{iff } p_i \models \phi_2 \text{ for some } i \geq 0, \text{ and} \\
 & p_j \models \phi_1 \text{ for any } j \text{ such that } 0 \leq j < i
 \end{array}$$

表 2.2: LTL の意味定義

## LTL の書き換え規則

LTL では、構文規則に表れる演算子の中に、表 2.3 のように他の演算子で書き換えができるものがある。これにより、LTL 式で使われる作用素を減らすことができる。

$$\begin{aligned}\phi_1 \vee \phi_2 &\equiv \neg(\neg\phi_1 \wedge \neg\phi_2) \\ \phi_1 \rightarrow \phi_2 &\equiv \neg\phi_1 \vee \phi_2 \\ G\phi &\equiv \neg F\neg\phi \\ F\phi &\equiv \text{true} U \phi\end{aligned}$$

表 2.3: LTL の書き換え規則の例

## LTL 式の例

図 2.1 は、LTL 式とそれを満たす経路の例である。

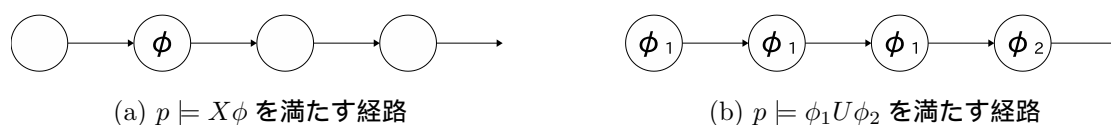


図 2.1: LTL 式の例

### 2.1.3 計算木論理 — CTL

計算木論理 (Computational Tree Logic, CTL) [3] とは、LTL の  $X, U$  などの時相演算子に加えて、経路に対する限量子  $E, A$  を導入した論理で、複数の経路に関する性質を記述できる論理である。

複数の経路を分岐した時系列と考えれば、CTL は分岐時間を扱う論理であるといえる。このように、分岐時間を扱う論理を、線形時間論理に対して分岐時間論理という。

#### CTL の構文

CTL の構文は表 2.4 の通りである。state-formula は、状態に関する式で状態式という。path-formula は、経路に関する式で経路式という。

$E, A$  は経路限量子であり、それぞれ、Exists, All という英単語に由来している。CTL の特徴として、経路限量子と時相演算子を組で用いる構文しか許されておらず、いくつかある分岐時間論理の中でも特に記述力が低い。

<i>CTL-formula</i>	::=	<i>state-formula</i>
<i>state-formula</i>	::=	<i>proposition</i>
		$\neg$ <i>state-formula</i>
		<i>state-formula</i> $\wedge$ <i>state-formula</i>
		<i>state-formula</i> $\vee$ <i>state-formula</i>
		<i>state-formula</i> $\rightarrow$ <i>state-formula</i>
		<i>E path-formula</i>
		<i>A path-formula</i>
<i>path-formula</i>	::=	<i>X state-formula</i>
		<i>F state-formula</i>
		<i>G state-formula</i>
		<i>state-formula U state-formula</i>

表 2.4: CTL の構文規則

### CTL の意味論

Kripke 構造  $M$  の状態  $s$  で状態式  $\phi$  が成り立つことを,  $M, s \models \phi$  と表す. 同様に, 経路  $p$  で経路式  $\psi$  が成り立つことを,  $M, p \models \psi$  と表す<sup>2</sup>.

CTL の時相演算子も, 直観的には LTL と同様の意味となる. CTL の正確な意味の定義は, 表 2.5 の通りである. ただし,  $s$  は状態,  $p$  は経路とする.

<b>state formula</b>	
$s \models \textit{proposition}$	iff $\textit{proposition} \in L(s)$
$s \models \neg\phi$	iff not $s \models \phi$
$s \models \phi_1 \wedge \phi_2$	iff $s \models \phi_1$ and $s \models \phi_2$
$s \models \phi_1 \vee \phi_2$	iff $s \models \phi_1$ or $s \models \phi_2$
$s \models E\psi$	iff $p \models \psi$ for some path $p = s \rightarrow s_1 \rightarrow \dots$
$s \models A\psi$	iff $p \models \psi$ for any path $p = s \rightarrow s_1 \rightarrow \dots$
<b>path formula</b> ( $p = s_0 \rightarrow s_1 \rightarrow \dots$ )	
$p \models X\phi$	iff $s_1$ exists and $s_1 \models \phi$
$p \models F\phi$	iff there exists a $i \geq 0$ such that $s_i \models \phi$
$p \models G\phi$	iff for any $i \geq 0$ , $s_i \models \phi$ and $s_{i+1}$ exists
$p \models \phi_1 U \phi_2$	iff $s_i \models \phi_2$ for some $i \geq 0$ , and $s_j \models \phi_1$ for any $j$ such that $0 \leq j < i$

表 2.5: CTL の意味定義

<sup>2</sup>LTL と同様,  $M$  を省略することもある.

## CTL の書き換え規則

CTL でも LTL と同様，構文規則に表れる演算子の中に，表 2.6 のように他の演算子で書き換えできるものがある．これにより，作用素の最小セットを構成することができ，その例としては  $\{false, \vee, \neg, EG, EU^3, EX\}$  などである．

$$\begin{aligned}
 EF \phi &\equiv E[true U \phi] \\
 AX \phi &\equiv \neg EX \neg \phi \\
 AG \phi &\equiv \neg EF \neg \phi \quad \equiv \neg E[true U \neg \phi] \\
 AF \phi &\equiv A[true U \phi] \quad \equiv \neg EG \neg \phi \\
 A[\phi_1 U \phi_2] &\equiv \neg(E[\neg \phi_2 U (\neg \phi_1 \wedge \neg \phi_2)] \vee EG \neg \phi_2)
 \end{aligned}$$

表 2.6: CTL の書き換え規則の例

## CTL 式の例

図 2.2 は，クリプキ構造上で CTL 式が成り立つ状態の集合を表したものである．

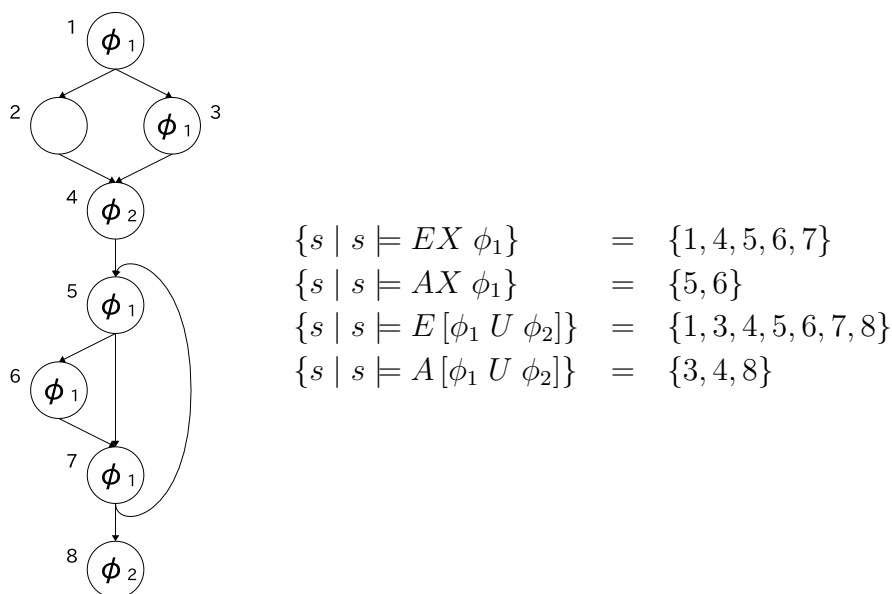


図 2.2: CTL 式の例

### 2.1.4 CTL\*

CTL\*[10] とは，LTL に経路限量子  $E$  と  $A$  を直接導入したものであり，CTL の「経路限量子と時相演算子は組になる必要がある」という制限がなくなった論理である．CTL\* は分岐時間論理である．

CTL\* の記述力は LTL と CTL を完全に包含する．簡単に言ってしまえば，CTL\* とは LTL と CTL を合わせたような論理と言える．

<sup>3</sup> $E[\dots U \dots]$  をこう略称する

## CTL\*の構文

CTL\*の構文は表 2.7 の通りである . CTL と同様 , *state-formula* は状態式 , *path-formula* は経路式である .

<i>CTL* -formula</i>	::=	<i>state-formula</i>
<i>state-formula</i>	::=	<i>proposition</i>
		$\neg$ <i>state-formula</i>
		<i>state-formula</i> $\wedge$ <i>state-formula</i>
		<i>state-formula</i> $\vee$ <i>state-formula</i>
		<i>state-formula</i> $\rightarrow$ <i>state-formula</i>
		<i>E path-formula</i>
		<i>A path-formula</i>
<i>path-formula</i>	::=	<i>CTL* -formula</i>
		$\neg$ <i>path-formula</i>
		<i>path-formula</i> $\wedge$ <i>path-formula</i>
		<i>path-formula</i> $\vee$ <i>path-formula</i>
		<i>path-formula</i> $\rightarrow$ <i>path-formula</i>
		<i>X path-formula</i>
		<i>F path-formula</i>
		<i>G path-formula</i>
		<i>path-formula U path-formula</i>

表 2.7: CTL\*の構文規則

## CTL\*の意味

CTL\*の正確な意味は , クリプキ構造  $M = (S, R, L)$  に対して , 表 2.8 のように定義される . ただし ,  $s$  は状態 ,  $p$  は経路とする . また ,  $p_i = s_i \rightarrow s_{i+1} \rightarrow \dots$  とする . 状態式の意味は CTL と同様であり , 経路式の意味は , LTL と同様である .

**state formula**

$s \models proposition$	iff	$proposition \in L(s)$
$s \models \neg\phi$	iff	not $s \models \phi$
$s \models \phi_1 \wedge \phi_2$	iff	$s \models \phi_1$ and $s \models \phi_2$
$s \models E\psi$	iff	$p \models \psi$ for some path $p = s \rightarrow s_1 \rightarrow \dots$
$s \models A\psi$	iff	$p \models \psi$ for any path $p = s \rightarrow s_1 \rightarrow \dots$

**path formula**

		$(p = s_0 \rightarrow s_1 \rightarrow \dots)$
$p \models \phi$	iff	$s_0 \models \phi$
$p \models \neg\psi$	iff	not $p \models \psi$
$p \models \psi_1 \wedge \psi_2$	iff	$p \models \psi_1$ and $p \models \psi_2$
$p \models \psi_1 \vee \psi_2$	iff	$p \models \psi_1$ or $p \models \psi_2$
$p \models X\psi$	iff	$p_1 \models \psi$
$p \models F\phi$	iff	there exists a $i \geq 0$ such that $s_i \models \phi$
$p \models G\psi$	iff	for any $i \geq 0$ , $p_i \models \psi$ and $p_{i+1}$ exists
$p \models \psi_1 U \psi_2$	iff	$p_i \models \psi_2$ for some $i \geq 0$ , and $p_j \models \psi_1$ for any $j$ such that $0 \leq j < i$

表 2.8: CTL\*の意味定義

**CTL\*の書き換え規則**

CTL\*も, LTL や CTL と同様, 他の演算子で書き換えできるものがある. LTL・CTL の書き換え規則がそのまま適用できる.

$$E\phi \equiv \neg A(\neg\phi)$$

表 2.9: CTL\*の書き換え規則の例

## 2.2 モデル検査

モデル検査とは、形式的検証手法の一つであり、状態遷移システムの状態空間を網羅的に探索することにより、システムの状態がある性質を満たすことを検証する手法である。

クリプキ構造  $M = (S, R, L)$  と満たすべき時相論理式  $\phi$  が与えられたとき、モデル検査を行うことで、ある状態  $s \in S$  が  $\phi$  を満たすかどうか、つまり

$$M, s \models \phi$$

が成り立つかどうかを検証できる。実際には、一つの状態  $s$  を固定して検査するのではなく、 $\phi$  を満たすような状態の集合

$$\{s \in S \mid M, s \models \phi\}$$

を求めて、この集合にある状態が入っているかどうかを調べる方法が一般的である。

## 2.3 NuSMV

モデル検査器の一つである NuSMV(New Symbolic Model Verifier)[11] は、シンボリックな手法を用いた最初のモデル検査器 SMV(Symbolic Model Verifier) を拡張したもので、扱える時相論理式は LTL と CTL の二つである。

二分決定グラフ ( Binary Decision Diagrams , BDD ) による手法と SAT ソルバを用いた限定モデル検査手法 [2] の両方を用いて検証を行うことができるという特徴をもつ。

### 2.3.1 モデル検査の概要

モデル検査の概要は、検査モデルの状態遷移関係を NuSMV 特有の言語で表したファイル (以下 smv ファイル) と検査式 (LTL または CTL) を入力し、その結果が真ならば true を、偽ならば false という結果とともに反例をひとつ表示する。NuSMV を用いてモデル検査を行う操作の流れを、図 2.3 に示す。

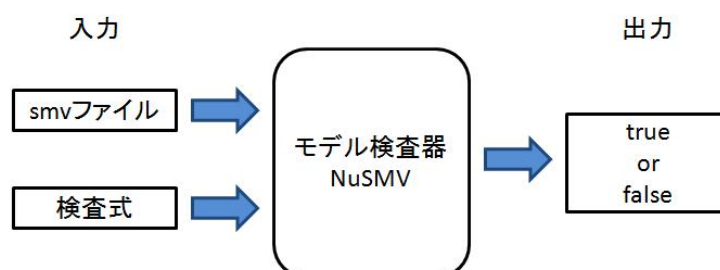


図 2.3: NuSMV の操作の流れ

## 2.3.2 smv ファイル

検査モデルの状態遷移関係を、変数の条件に着目して表したもの。SMV 及び NuSMV において有限状態遷移系を定義する場合、状態を変数の組みで表現し、遷移関係状態と状態の直積上の関係で与える。例えば、状態を構成する 2 変数を  $x, y$  とし、それぞれ  $0, 1, 2$  の値をとるものとする。このとき、各状態は

$$(x, y) = (0, 0), (0, 1), (0, 2), (1, 0), \dots$$

のように表現される。状態遷移関係を図 2.4 であるとする。このときの smv ファイルを次に示す。

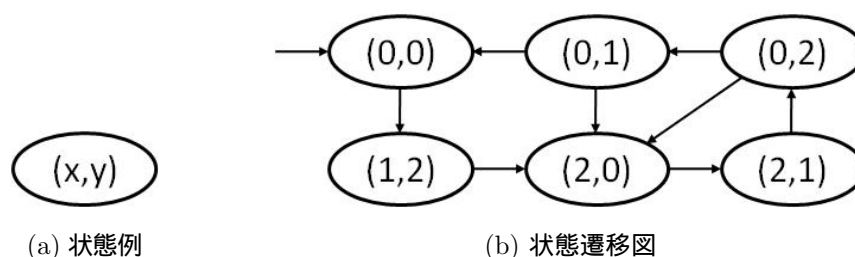


図 2.4: 例の状態遷移図

smv ファイルの仕様はいくつかの部分から構成されており、「MODULE」でモジュール宣言を開始する。main モジュールはすべての smv ファイルに必ず 1 つ含まれており、main から参照可能な変数の組を NuSMV での状態とする。

図 2.4(b) の smv ファイル

```

1:MODULE main
2:VAR
3:  x : {0,1,2};
4:  y : {0,1,2};
5:ASSIGN
6:  init(x) := 0;
7:  init(y) := 0;
8:
9:  next(x) :=
10: case
11:   x = 0 & y = 0 : 1;
12:   x = 0 & y = 1 : {0,2};
13:   x = 0 & y = 2 : {0,2};
14:   x = 1 & y = 2 : 2;
15:   x = 2 & y = 0 : 2;
16:   1                : 0;
17: esac;
18:
19:  next(y) :=
20: case
21:   x = 0 & y = 0 : 2;
22:   x = 0 & y = 1 : 0;
23:   x = 0 & y = 2 : {0,1};
24:   x = 1 & y = 2 : 0;
25:   x = 2 & y = 0 : 1;
26:   1                : 2;
27: esac;
28:
29: LTLSPEC G !(x = 2 & y = 2)
30: CTLSPEC AF (x = 2 & y = 0)
31: CTLSPEC EF (x = 1 & y = 0)

```

- VAR : 2-4 行部分

「VAR」により変数宣言モードとなる。この例では2変数からなる状態空間となる。それぞれの変数は{と}の間に列挙された値をとることができる。

- ASSIGN : 5-27 行部分

「ASSIGN」以降に遷移規則を記述する「init(変数名) := 値;」で初期状態を、「next(変数名) := 値;」で変数の遷移後の値を定義する。

6-7行目により初期状態  $(x, y) = (0, 0)$  が設定される。9-17行により、変数  $x$  についての遷移規則を記述している。 $x$  の遷移規則は case 文により条件分けされている。case 文の中は「条件:値」という形をしており、条件を満たす時に値を取ることがで

きる．12 行目のように値を集合により与えると，集合の中から 1 つを次の値として非決定的に選択する．この場合では，0 か 2 が非決定に選択される．

- SPEC : 29-31 行部分

「LTLSPEC」もしくは「CTLSPEC」の後に検証する検査式を記述する「LTLSPEC」の後の検査式は LTL に、「CTLSPEC」の後の検査式は CTL に従った時相論理を用いて表す．

29 行目は「 $x = 2$  かつ  $y = 2$  となる状態には常に到達しない」という性質を表している．

30 行目は「どんな経路でもいつか  $x = 2$  かつ  $y = 0$  となる状態に到達する」という性質を表している．

31 行目は「いつか  $x = 1$  かつ  $y = 0$  となる状態に到達する経路が存在する」という性質を表している．

### 2.3.3 NuSMV での LTL 検査

分岐を含んだモデルに対する LTL 検査については，LTL は単一の経路に関する性質を表したものであるため，初期状態から辿ることができる全ての経路に関して，それぞれ LTL 検査式を検査し，その全てが真ならば true を，そうでなければ false という結果と共に，反例となる経路をひとつ表示する．つまり，NuSMV で LTL 式  $\phi$  を検査する場合， $A\phi$  という CTL\* 式を検査することと原理的に等しいことである．

## 2.4 Buchi オートマトン

Buchi オートマトンとは、受理状態を無限回訪れるような無限長の語 (ラベル列) の集合が受理言語であるようなオートマトンである。そもそも有限オートマトンの受理言語は、初期状態から受理状態を導く有限長の語  $a_0a_1\dots a_n$  の集合である。このような受理言語は、受理状態で停止するようなシステムであればその動作を表現するのに適しているが、常に外部からの要求に応答し続けるインタラクティブシステムのように、停止しないことが重要になるシステムには適していない。そのようなシステムの動作を表現するのに適したオートマトンの1つが Buchi オートマトンである。

### 2.4.1 定義

Buchi オートマトンは有限オートマトンと同じ5つ組み  $(S, S_0, \Sigma, \delta, F)$  で定義される。

- $S$ : 非空な状態の有限集合
- $S_0$ : 非空な初期状態の集合
- $\Sigma$ : アルファベットの有限集合
- $\delta: S \times \Sigma \times S$ : 遷移の集合
- $F$ : 受理状態の集合

しかし、Buchi オートマトンは無限長の語や実行を扱うため、その受理の考え方が異なる。

無限長のラベルの列  $a_0a_1\dots a_i\dots$  が Buchi オートマトン  $B = (S, S_0, \Sigma, \delta, F)$  の語であるとは、次の条件を満たす状態  $s_0, s_1, \dots, s_i, s_{i+1}, \dots$  が存在することである。

$$(s_0, a_0, s_1), (s_1, a_1, s_2), \dots, (s_i, a_i, s_{i+1}), \dots \in \delta$$

ここで、 $s_0 \in S_0$  は初期状態であり、上の状態の列  $s_0 \cdot s_1 \cdots s_i \cdot s_{i+1} \cdots$  をこの Buchi オートマトン  $B$  の実行という。また、 $B$  の実行である遷移列の中にある受理状態  $s \in F$  が無限回含まれているならば、その無限長の語と実行は  $B$  によって受理されるという。 $s$  が無限回含まれるとは、任意の  $i \geq 0$  について、 $s = s_j$  となるある  $j \geq i$  が存在することである。Buchi オートマトン  $B$  によって受理される全ての無限長の語の集合 (受理言語) を  $Lang_\omega(B)$ 、受理される全ての無限長の実行の集合を  $Run_\omega(B)$  で表すとき、これらの詳しい定義は以下のようなになる。ただし、実行  $\sigma = s_0 \cdot s_1 \cdots$  に対して  $lim(\sigma)$  を、実行  $\sigma$  に無限回表れる全ての状態の集合とする。

$$\begin{aligned} Lang_\omega(B) &= \{w \mid w \text{ は } B \text{ によって受理される語}\} \\ &= \{a_0 \cdot a_1 \cdots \mid \exists s_0, s_1, \dots \\ &\quad (s_0 \in S_0 \wedge (lim(s_0 \cdot s_1 \cdots) \cap F \neq \emptyset) \wedge (\forall i \geq 0((s_i, a_i, s_{i+1}) \in \delta)))\} \end{aligned}$$

$$\begin{aligned}
Ran_\omega(B) &= \{ \sigma \mid \sigma \text{ は } B \text{ によって受理される実行} \} \\
&= \{ s_0 \cdot s_1 \cdots \mid \exists a_0, a_1, \dots \\
&\quad (s_0 \in S_0 \wedge (lim(s_0 \cdot s_1 \cdots) \cap F \neq \emptyset) \wedge (\forall i \geq 0((s_i, a_i, s_{i+1}) \in \delta))) \}
\end{aligned}$$

## 2.4.2 拡張 Buchi オートマトン

Buchi オートマトン  $B = (S, S_0, \Sigma, \delta, F)$  の受理状態集合  $F \in S$  を, 受理状態集合の集合  $\Gamma = \{F_1, \dots, F_n\} \subseteq 2^S$  に拡張した Buchi オートマトンを拡張 Buchi オートマトンという. 拡張 Buchi オートマトン  $G = (S, S_0, \Sigma, \delta, \Gamma)$  によって実行  $\sigma = s_0 \cdot s_1 \cdots$  が受理されるとは, 各  $F \in \Gamma$  について,  $lim(\sigma) \cap F \neq \emptyset$  が成り立つことである. 同様に, 受理される語とは, 受理される実行を導くラベル列である. 拡張 Buchi オートマトン  $G$  によって受理される語と実行の集合の詳しい定義を以下に示す.

$$\begin{aligned}
Lang_\omega(G) &= \{ w \mid w \text{ は } G \text{ によって受理される語} \} \\
&= \{ a_0 \cdot a_1 \cdots \mid \exists s_0, s_1, \dots \\
&\quad ((s_0 \in S_0) \wedge (\forall F \in \Gamma((lim(s_0 \cdot s_1 \cdots) \cap F \neq \emptyset)) \wedge (\forall i \geq 0((s_i, a_i, s_{i+1}) \in \delta)))) \} \\
Ran_\omega(G) &= \{ w \mid w \text{ は } G \text{ によって受理される実行} \} \\
&= \{ s_0 \cdot s_1 \cdots \mid \exists a_0, a_1, \dots \\
&\quad ((s_0 \in S_0) \wedge (\forall F \in \Gamma((lim(s_0 \cdot s_1 \cdots) \cap F \neq \emptyset)) \wedge (\forall i \geq 0((s_i, a_i, s_{i+1}) \in \delta)))) \}
\end{aligned}$$

定義から明らかであるように, もし受理状態集合  $\Gamma$  が空集合であれば, 全ての語や実行が受理されることになる<sup>4</sup>. すなわち, このような拡張 Buchi オートマトンは, 全ての状態が受理状態である Buchi オートマトン  $G = (S, S_0, \Sigma, \delta, S)$  と同じである.

拡張 Buchi オートマトン  $G = (S, S_0, \Sigma, \delta, \{F_1, \dots, F_n\})$  を, それと受理言語が等しい通常の Buchi オートマトン  $B = GBAtaBA(G) = (S', S'_0, \Sigma', \delta', F')$  に変換する関数  $GBAtaBA$  は次のように与えられる.

- $n = 0$  の場合  
 $S' = S, S'_0 = S_0, \Sigma' = \Sigma, \delta' = \delta, F' = F$

- $n \geq 1$  の場合

$$S' = S \times \{1, 2, \dots, n\}$$

$$S'_0 = S_0 \times \{1\}$$

$$\Sigma' = \Sigma$$

$$\begin{aligned}
\delta' &= \{((s, i), a, (s', i) \mid (s, a, s') \in \delta \wedge s \notin F_i\} \cup \\
&\quad \{((s, i), a, (s', i+1) \mid (s, a, s') \in \delta \wedge s \in F_i \wedge i < n\} \cup \\
&\quad \{((s, n), a, (s', 1) \mid (s, a, s') \in \delta \wedge s \in F_n\}
\end{aligned}$$

$$F = F_1 \times \{1\}$$

<sup>4</sup> $\forall \Gamma \in \emptyset \dots$  は常に真

この  $n \geq 1$  の場合の変換方法では、各状態を受理状態集合  $F_i$  の数  $n$  だけ複製し、 $(s, a, s') \in \delta$  ならば、 $(s, i)$  の  $a$  による遷移先を  $(s', i')$  にする。ここで、インデックス  $i'$  は、 $s \notin F_i$  ならば  $i = i'$  であり、 $s \in F_i$  ならば  $i + 1$  ( $n$  を超えた時は  $1$  にもどる) である。これによって、 $GBAtoBA(G)$  の受理状態集合では  $F_1$  のみ考慮されるが、 $F_1$  の受理状態を無限回訪されるためには、各  $F_2, \dots, F_n$  の受理集合も無限回訪れることが必要になり、元の拡張 Buchi オートマトンと同じ受理言語が得られる。

この拡張 Buchi オートマトンは、後に説明する LTL モデル検査のアルゴリズムにおいて LTL 式を Buchi オートマトンに変換するときに使われる。

### 2.4.3 Buchi 受理言語の和集合，積集合，補集合

2 つの Buchi オートマトン  $B_1 = (S_1, S_{01}, \sum_1, \delta_1, F_1)$ 、 $B_2 = (S_2, S_{02}, \sum_2, \delta_2, F_2)$  が与えられたとき、それらの Buchi 受理言語の和集合  $Lang_\omega(B_1) \cup Lang_\omega(B_2)$ 、積集合  $Lang_\omega(B_1) \cap Lang_\omega(B_2)$ 、補集合  $\overline{Lang_\omega(B_1)}$  を受理言語にもつ Buchi オートマトンの合成法を以下に示す。ただし、ここでは一般性を失うことなく  $S_1 \cap S_2 = \emptyset$  と定義する。

- 和集合

Buchi オートマトンについても

$$Lang_\omega(B_1 \cup B_2) = Lang_\omega(B_1) \cup Lang_\omega(B_2)$$

が成り立つ。ここで  $B_1 \cup B_2$  は有限オートマトンと同じ定義。

- 積集合

和集合と同様に  $B_1 \times B_2$  を有限オートマトンと同じ定義とすると、

$$Lang_\omega(B_1 \times B_2) = Lang_\omega(B_1) \cap Lang_\omega(B_2)$$

ではなく、

$$Lang_\omega(B_1 \times B_2) \subseteq Lang_\omega(B_1) \cap Lang_\omega(B_2)$$

が成り立つ。ここで拡張 Buchi オートマトン  $B_1 \times_G B_2 = (S, S_0, \sum, \delta, \{F'_1, F'_2\})$  を次のように定義する。

$$S = S_1 \times S_2 = \{(s_1, s_2) \mid s_1 \in S_1 \wedge s_2 \in S_2\}$$

$$S_{01} \times S_{02}$$

$$\sum = \sum_1 \cap \sum_2$$

$$\delta = \{((s_1, s_2), a, (s'_1, s'_2)) \mid (s_1, a, s'_1) \in \delta_1 \wedge (s_2, a, s'_2) \in \delta_2\}$$

$$F'_1 = \{(s_1, s_2) \mid s_1 \in F_1 \wedge s_2 \in S_2\}$$

$$F'_2 = \{(s_1, s_2) \mid s_1 \in S_1 \wedge s_2 \in F_2\}$$

拡張 Buchi オートマトン  $B_1 \times_G B_2$  によって受理される語は、 $B_1$  と  $B_2$  の両方の受理状態を無限回訪れなければならないが、 $B_1 \times_G B_2$  の場合と異なり同時に訪れる必要はない。このとき、

$$Lang_\omega(B_1 \times_G B_2) = Lang_\omega(B_1) \cap Lang_\omega(B_2)$$

が成り立つ．既に，前の節で任意の拡張 Buchi オートマトンを Buchi オートマトンに変換する関数  $GBAtoBA(G)$  は与えられているので，Buchi オートマトン  $B_1 \cap B_2$  を  $GBAtoBA(B_1 \times_G B_2)$  と定義すれば，

$$\begin{aligned}Lang_{\omega}(B_1 \cap B_2) &= Lang_{\omega}(GBAtoBA(B_1 \times_G B_2)) \\ &= Lang_{\omega}(B_1 \times_G B_2) \\ &= Lang_{\omega}(B_1) \times_G Lang_{\omega}(B_2)\end{aligned}$$

が成り立つ．

- 補集合  
任意の Buchi オートマトンの受理言語が与えられたとき，その補集合に相当する受理言語をもつ Buchi オートマトンは存在するが，ここではその合成法は省略する．詳しくは文献 [15] を参照のこと．

# 第3章 CTL\*モデル検査器の設計と実装

本章では、CTL\*のモデル検査アルゴリズムと、モデル検査器を実装したプログラムについて説明する。

## 3.1 アルゴリズム

CTL\*モデル検査のアルゴリズムを説明するために、LTLモデル検査・CTLモデル検査について説明する。

### 3.1.1 LTLモデル検査のアルゴリズム

検査対象のモデルとLTL検査式を否定した検査式をそれぞれ、Buchiオートマトン [18] といわれる無限実行列を受理できるようなオートマトンに変換し、その二つの同期積をとったBuchiオートマトンが受理する実行の集合が、空であるときモデル検査の結果はtrueと、そうでないときfalseと判定する。アルゴリズムの概要を図3.1に表す。

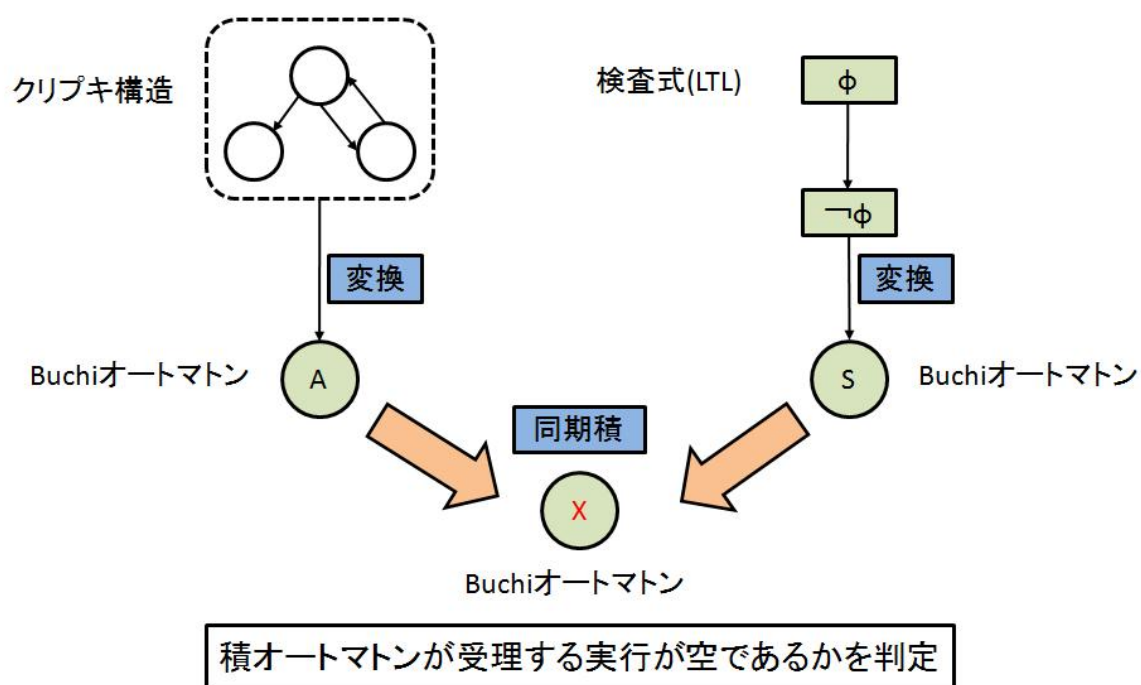


図 3.1: LTLモデル検査アルゴリズムの概略図

### 3.1.2 CTL モデル検査のアルゴリズム

CTL モデル検査は、与えられた式を部分式に分けて検査していく [5]。CTL 検査式  $f$  を構成する全ての部分式の集合を考え、モデルの状態遷移図の各状態  $s$  に、 $s$  において真である部分式の集合  $label(s)$  をラベル付ける。

例として、検査式  $f = E[true U (p \wedge EX(\neg p))]$  のモデル検査を考える。以下のように、1 番小さい部分式 ( $level\ 0$ ) からボトムアップで集合を考えていくことで、すでに検査した部分式を原始命題と同じように扱うことができる。

$level\ 0$   $p, \neg p$  のラベル付け

$level\ 1$   $EX(\neg p)$  のラベル付け

$level\ 2$   $p \wedge EX(\neg p)$  のラベル付け

$level\ 3$   $E[true U (p \wedge EX(\neg p))]$  のラベル付け

具体的なラベル付けのアルゴリズムを Algorithm3.1 ~ 3.4 に示す。ただし、 $S$  は状態の集合、 $label(s)$  はラベリング関数とする。

時相演算子や経路限量子を含んだ CTL 式に対して、その全ては書き換え規則により  $EX, AF, EU$  に書き換え可能である。よって、 $EX\ g, AF\ g, E[g\ U\ h]$  のそれぞれに対して真になるような状態の集合を用意すれば、全ての CTL に対するラベル付けができる。具体的な  $EX\ g, AF\ g, E[g\ U\ h]$  のラベル付けのアルゴリズムが Algorithm3.2 ~ 3.4 である。ただし、 $pre_{\exists}(X), pre_{\forall}(X)$  を以下のように定義する。

$$pre_{\exists}(X) = \{s \in S \mid \exists s', s \rightarrow s' \text{ and } s' \in X\}$$

$$pre_{\forall}(X) = \{s \in S \mid \forall s', s \rightarrow s' \text{ implies } s' \in X\}$$

---

**Algorithm 3.1**  $g$  のラベル付け

---

**SAT**( $g$ )

```
1: begin
2:   case
3:      $g$  is  $\top$  : retrun  $S$ 
4:      $g$  is  $\perp$  : retrun  $\emptyset$ 
5:      $g$  is atomic : retrun  $\{s \in S \mid g \in \text{label}(s)\}$ 
6:      $g$  is  $\neg g_1$  : retrun  $S - \text{SAT}(g_1)$ 
7:      $g$  is  $g_1 \wedge g_2$  : retrun  $\text{SAT}(g_1) \cap \text{SAT}(g_2)$ 
8:      $g$  is  $g_1 \vee g_2$  : retrun  $\text{SAT}(g_1) \cup \text{SAT}(g_2)$ 
9:      $g$  is  $g_1 \rightarrow g_2$  : retrun  $\text{SAT}(\neg g_1 \vee g_2)$ 
10:     $g$  is  $AX\ g_1$  : retrun  $\text{SAT}(\neg EX \neg g_1)$ 
11:     $g$  is  $EX\ g_1$  : retrun  $\text{SAT}_{EX}(g_1)$ 
12:     $g$  is  $AF\ g_1$  : retrun  $\text{SAT}_{AF}(g_1)$ 
13:     $g$  is  $EF\ g_1$  : retrun  $\text{SAT}(E[\top U\ g_1])$ 
14:     $g$  is  $AG\ g_1$  : retrun  $\text{SAT}(\neg EF \neg g_1)$ 
15:     $g$  is  $EG\ g_1$  : retrun  $\text{SAT}(\neg AF \neg g_1)$ 
16:     $g$  is  $A[g_1 U\ g_2]$  : retrun  $\text{SAT}(\neg(E[\neg g_2 U(\neg g_1 \wedge \neg g_2)]) \vee EG \neg g_2)$ 
17:     $g$  is  $E[g_1 U\ g_2]$  : retrun  $\text{SAT}_{EU}(g_1, g_2)$ 
18:   end case
19: end function
```

---

---

**Algorithm 3.2**  $EX\ g$  のラベル付け

---

**SAT**<sub>EX</sub>( $g$ )

```
1: local var  $X, Y$ 
2: begin
3:    $X := \text{SAT}(g)$ ;
4:    $Y := \text{pre}_{\exists}(X)$ ;
5:   return  $Y$ ;
6: end
```

---

---

**Algorithm 3.3**  $AF$   $g$  のラベル付け

---

**SAT**<sub>AF</sub>( $g$ )

```
1: local var  $X, Y$ 
2: begin
3:   do
4:      $X := S$ ;
5:      $Y := \text{SAT}(X)$ ;
6:   while( $X = Y$ )
7:   begin
8:      $X := Y$ ;
9:      $Y := Y \cup \text{pre}_\forall(Y)$ ;
10:  end
11:  return  $Y$ ;
12: end
```

---

---

**Algorithm 3.4**  $E[g \cup h]$  のラベル付け

---

**SAT**<sub>EU</sub>( $g, h$ )

```
1: local var  $W, X, Y$ 
2: begin
3:   do
4:     SAT( $g$ );
5:      $X := S$ ;
6:      $Y := \text{SAT}(h)$ ;
7:   while( $X = Y$ )
8:   begin
9:      $X := Y$ ;
10:     $Y := Y \cup (W \cap \text{pre}_\forall(Y))$ ;
11:  end
12:  return  $Y$ ;
13: end
```

---

### 3.1.3 CTL\*モデル検査のアルゴリズム

CTL\*モデル検査はLTLとCTLのモデル検査を使ったものとなる [4] . 基本的にはCTLモデルチェックと同様に, CTL\*式に対してそれを構成する最小の部分式を検査し, その部分式を原始命題とみなす操作を繰り返し, 与えられた CTL\*式が部分式に分解できなくなるまで行う. 具体的なアルゴリズムを次に示す.

---

**Algorithm 3.5**  $Ag$  のラベル付け

---

labelA( $g$ )

```
1: begin
2: if  $g$  is a CTL formula then
3:   apply CTL model checking for  $g$ ;
4:   return ;
5:  $g' := g[a_1/\mathbf{A}h_1, \dots, a_k/\mathbf{A}h_k]$ ;
6: for all  $s \in S$  do
7:   for  $i = 1, \dots, k$  do
8:     if  $\mathbf{A}h_i \in \text{label}(s)$  then
9:        $\text{label}(s) := \text{label}(s) \cup \{a_i\}$ ;
10:  apply LTL model checking for  $g'$ ;
11:  for all  $s \in S$  do
12:    if  $g' \in \text{label}(s)$  then
13:       $\text{label}(s) := \text{label}(s) \cup \{g\}$ ;
14: end function
```

---

- 1-2 行

まず, 検査式の中の経路限量子  $E$  を書き換え規則に従って  $A$  に書き換える<sup>1</sup>. その検査式  $g$  が CTL ならば CTL モデル検査を適用する.

- 4 行

検査式  $g$  が CTL\*特有の形であるとき,  $g$  の部分式  $\mathbf{A}h_i$  を原始命題  $a_i$  に置き換える. この作業を最も小さい部分式からボトムアップで行っていき, 部分式が無くなるまで置き換えた式を  $g'$  とする.

- 6-8 行

次に, 置き換えていった部分式  $\mathbf{A}h_i$  全てに対して, 各状態でのラベリング関数の更新を行う.

- 9-12 行

最終的な検査式  $g'$  にモデル検査を行い, その結果を使って各状態での検査式  $g$  のラベリング関数の更新を行う.

---

<sup>1</sup> $E \rightarrow A$  の書き換え規則は  $E\phi \equiv \neg A(\neg\phi)$

## 3.2 実装

ここでは、CTL\*のモデル検査アルゴリズムを元の実装を行ったシステムの解説を行う。

### 3.2.1 システム

本システムの入力は以下の3つである。

- smv\*ファイル  
内容はNuSMVの入力ファイルと同じ。変数宣言をするVAR部分とASSIGN部分で初期値宣言をするinit部分、同じくASSIGN部分で変数の変化条件を表すnext部分をそれぞれVARファイル、INITファイル、NEXTファイルの3つに分けて入力する。この3つのファイルをまとめてsmv\*ファイルと呼ぶ。
- 検査式  
CTL\*の検査式
- モデルのクリプキ構造  
検査対象であるモデルのクリプキ構造の情報。詳しくは、各状態の変数の値と遷移関係を表したデータ構造。

本システムの概略図を図3.2に示す。

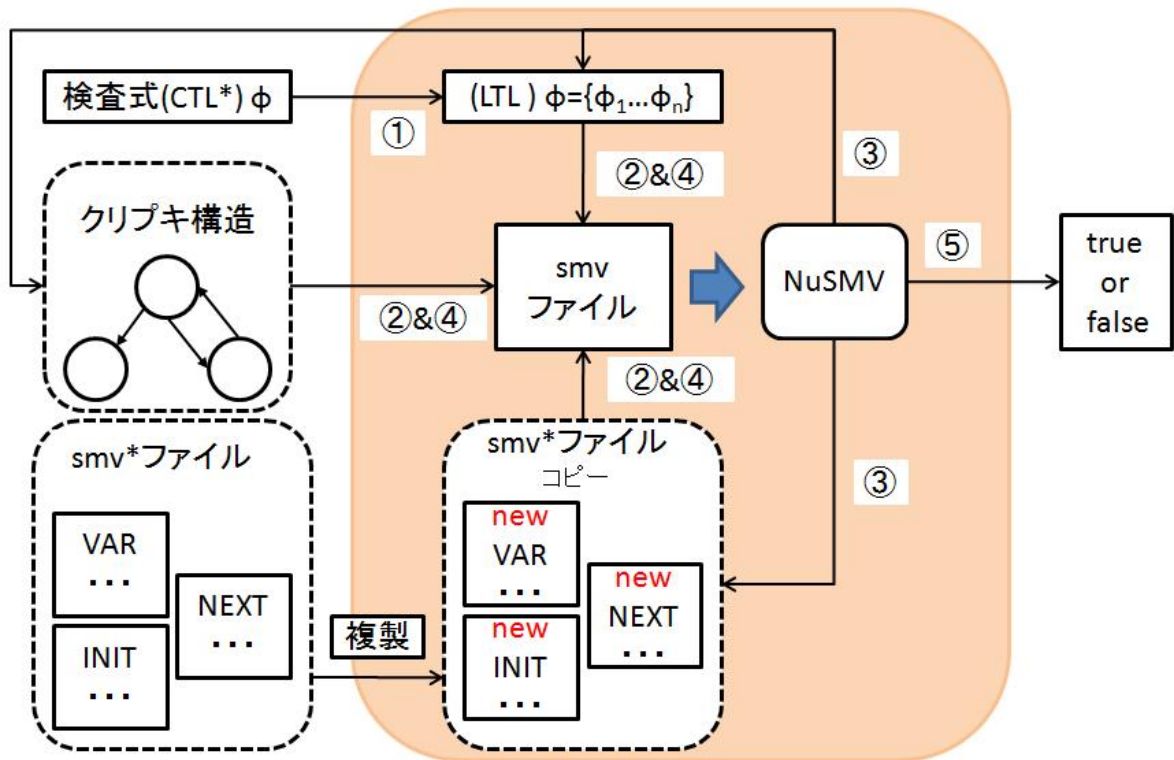


図 3.2: 本システムの概略図

### 検査式の分解

与えられた検査式に対して，経路限量子の  $E$  を  $A$  に変換し，CTL\*の部分式に分解する．

### NuSMV での検査

入力された smv\*ファイルは3つとも複製をつくり，以後 smv\*ファイルの更新にはこのファイルを使用する．分解した部分式の中で最も小さい式を，入力されたクリプキ構造の各状態について検査していく．その際，INIT ファイルを更新しながら，検査式と合わせて NuSMV 入力用の smv ファイルを作成し，NuSMV によって検査していく．

### 検査結果を更新

の結果をもとに検査した式を新たな原始命題  $x_i$  とみなして，VAR ファイル，NEXT ファイル，モデルのクリプキ構造， で分解した検査式をそれぞれ更新する．

### 再帰

検査式の中で次に大きな部分式を使って，検査式が原始命題のみになるまで を繰り返す．

### 結果出力

最終的な結果を出力．

## 3.3 CTL\*モデル検査の例

CTL\*モデル検査器のシステムを，例を使って説明する．

### 3.3.1 例. 電子レンジ

本システムの説明に使用する，電子レンジのモデルの状態遷移図を図 3.3 に示す．このモデルに対して CTL\*検査式  $AG ((!Close \ \& \ Start) \rightarrow A(G \ !Heat \ | \ F \ !Error))$  を検査する．この式は「任意の状態において扉を閉めずにスタートするならば，それ以降のどの状態でも温められないか，もしくは将来エラーから復帰する」ということを表したもので，LTL でも CTL でも扱えない式である．<sup>2</sup>

このモデルに対する，smv\*ファイル (VAR ファイル，INIT ファイル，NEXT ファイル) を次に示す．

<sup>2</sup>検査式において， $!$ ， $\&$ ， $|$  はそれぞれ  $\neg$ ， $\wedge$ ， $\vee$  の意味で，NuSMV 特有のもの．

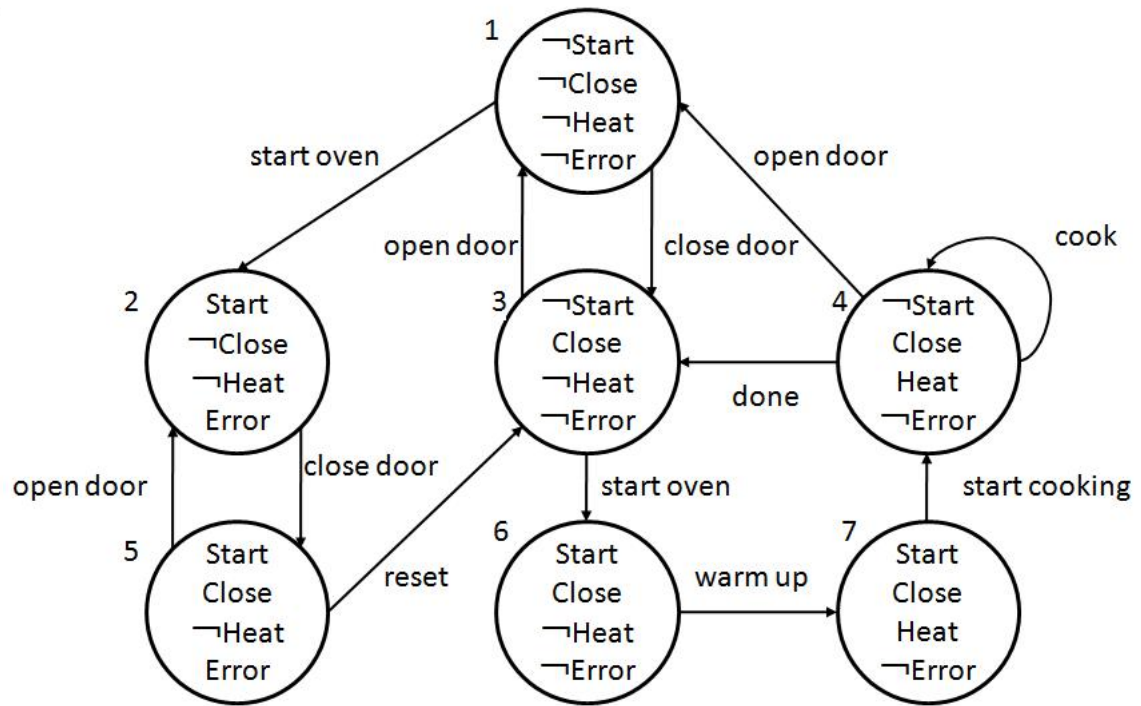


図 3.3: 電子レンジの状態遷移図

VAR ファイル

```

1:MODULE main
2:VAR
3: Start : {0,1};
4: Close : {0,1};
5: Heat : {0,1};
6: Error : {0,1};

```

INIT ファイル

```

1:ASSIGN
2: init(Start) := 0;
3: init(Close) := 0;
4: init(Heat) := 0;
5: init(Error) := 0;

```

## NEXT ファイル

```
1: next(Start) :=
2: case
3:   Start = 0 & Close = 0 & Heat = 0 & Error = 0: {0,1};
4:   Start = 1 & Close = 0 & Heat = 0 & Error = 1: 1;
5:   Start = 0 & Close = 1 & Heat = 0 & Error = 0: {0,1};
6:   Start = 0 & Close = 1 & Heat = 1 & Error = 0: 0;
7:   Start = 1 & Close = 1 & Heat = 0 & Error = 1: {0,1};
8:   Start = 1 & Close = 1 & Heat = 0 & Error = 0: 1;
9:   1 : 0;
10: esac;
11: next(Close) :=
12: case
13:   Start = 0 & Close = 0 & Heat = 0 & Error = 0: {0,1};
14:   Start = 1 & Close = 0 & Heat = 0 & Error = 1: 1;
15:   Start = 0 & Close = 1 & Heat = 0 & Error = 0: {0,1};
16:   Start = 0 & Close = 1 & Heat = 1 & Error = 0: {0,1};
17:   Start = 1 & Close = 1 & Heat = 0 & Error = 1: {0,1};
18:   Start = 1 & Close = 1 & Heat = 0 & Error = 0: 1;
19:   1 : 1;
20: esac;
21: next(Heat) :=
22: case
23:   Start = 0 & Close = 0 & Heat = 0 & Error = 0: 0;
24:   Start = 1 & Close = 0 & Heat = 0 & Error = 1: 0;
25:   Start = 0 & Close = 1 & Heat = 0 & Error = 0: 0;
26:   Start = 0 & Close = 1 & Heat = 1 & Error = 0: {0,1};
27:   Start = 1 & Close = 1 & Heat = 0 & Error = 1: 0;
28:   Start = 1 & Close = 1 & Heat = 0 & Error = 0: 1;
29:   1 : 1;
30: esac;
31: next(Error) :=
32: case
33:   Start = 0 & Close = 0 & Heat = 0 & Error = 0: {0,1};
34:   Start = 1 & Close = 0 & Heat = 0 & Error = 1: 1;
35:   Start = 0 & Close = 1 & Heat = 0 & Error = 0: 0;
36:   Start = 0 & Close = 1 & Heat = 1 & Error = 0: 0;
37:   Start = 1 & Close = 1 & Heat = 0 & Error = 1: {0,1};
38:   Start = 1 & Close = 1 & Heat = 0 & Error = 0: 0;
39:   1 : 0;
40: esac;
```

検査の過程を次に示す．

#### step1 検査式の分解

検査式  $AG ((!Close \ \& \ Start) \rightarrow A(G !Heat \ | \ F !Error))$  を部分式に分解する．  
部分式は小さい順に示すと次のようになる．ただし，*level 1* での記号 *level 0* は原始命題．

*level 0*  $A(G !Heat \ | \ F !Error)$

*level 1*  $AG ((!Close \ \& \ Start) \rightarrow level \ 0)$

#### step2 $A(G !Heat \ | \ F !Error)$ の検査

入力された smv\* ファイルを複製する．以後，smv\* ファイルの更新はこの複製に対して行う．次に，電子レンジのモデルの各状態に対して， $A(G !Heat \ | \ F !Error)$  の検査を行う．

具体的には，INIT ファイルを更新しながら，3 つの smv\* ファイルと検査式「LTLSPEC  $G !Heat \ | \ F !Error$ 」を合わせて，NuSMV 入力用の smv ファイルを作成する．

#### step3 検査結果を更新

step2 での結果で smv\* ファイル，クリプキ構造，検査式を更新する．その際， $A(G !Heat \ | \ F !Error)$  は原始命題  $x_0$  と置き換える．検査式の更新結果は  $AG ((!Close \ \& \ Start) \rightarrow x_0)$  となる．  
smv\* ファイルの更新結果は  $x_0$  の更新に，クリプキ構造の更新結果を図 3.4 に示す．

#### step4 $AG ((!Close \ \& \ Start) \rightarrow x_0)$ の検査

step2 と同様に  $AG ((!Close \ \& \ Start) \rightarrow x_0)$  の検査をし，step3 と同様に結果を更新する．検査をする際の検査式は「LTLSPEC  $G((!Close \ \& \ Start) \rightarrow x_0)$ 」となる<sup>3</sup>．

#### step5 検査結果を更新

step3 と同様に smv\* ファイル，クリプキ構造，検査式を更新する．  
その際， $AG ((!Close \ \& \ Start) \rightarrow x_0)$  は原始命題  $x_1$  と置き換える．検査式の更新結果は  $x_1$  となる．

#### step6 $x_1$ の検査

step2 と同様に  $x_1$  の検査をする．検査をする際の検査式は「LTLSPEC  $x_1$ 」となる．その結果を最終的な答えとして出力する．

以上の過程から，本システムは電子レンジのモデルに対する CTL\* の検査式  $AG ((!Close \ \& \ Start) \rightarrow A(G !Heat \ | \ F !Error))$  の検査結果を true と出力する．

<sup>3</sup>NuSMV で LTL 検査式  $\phi$  を検査する場合，原理的には  $A\phi$  という CTL\* 式を検査していることと等しい．詳しくは 2.3.3 を参照

$x_0$  の更新をした VAR ファイル

```
1:MODULE main
2:VAR
3: Start : {0,1};
4: Close : {0,1};
5: Heat  : {0,1};
6: Error : {0,1};
7: x0    : {0,1};
```

$x_0$  の更新をした INIT ファイル

```
1:ASSIGN
2:  init(Start) := 0;
3:  init(Close) := 0;
4:  init(Heat)  := 0;
5:  init(Error) := 0;
6:  init(x0)    := 1;
```

$x_0$  の更新をした NEXT ファイル

```
1:  next(Start) :=
    ...
11: next(Close) :=
    ...
21: next(Heat) :=
    ...
31: next(Error) :=
    ...
41: next(x0) :=
42: case
43:   Start = 0 & Close = 0 & Heat = 0 & Error = 0: 1;
44:   Start = 1 & Close = 0 & Heat = 0 & Error = 1: 1;
45:   Start = 0 & Close = 1 & Heat = 0 & Error = 0: 1;
46:   Start = 0 & Close = 1 & Heat = 1 & Error = 0: 1;
47:   Start = 1 & Close = 1 & Heat = 0 & Error = 1: 1;
48:   Start = 1 & Close = 1 & Heat = 0 & Error = 0: 1;
49:   1 : 1;
50: esac;
```

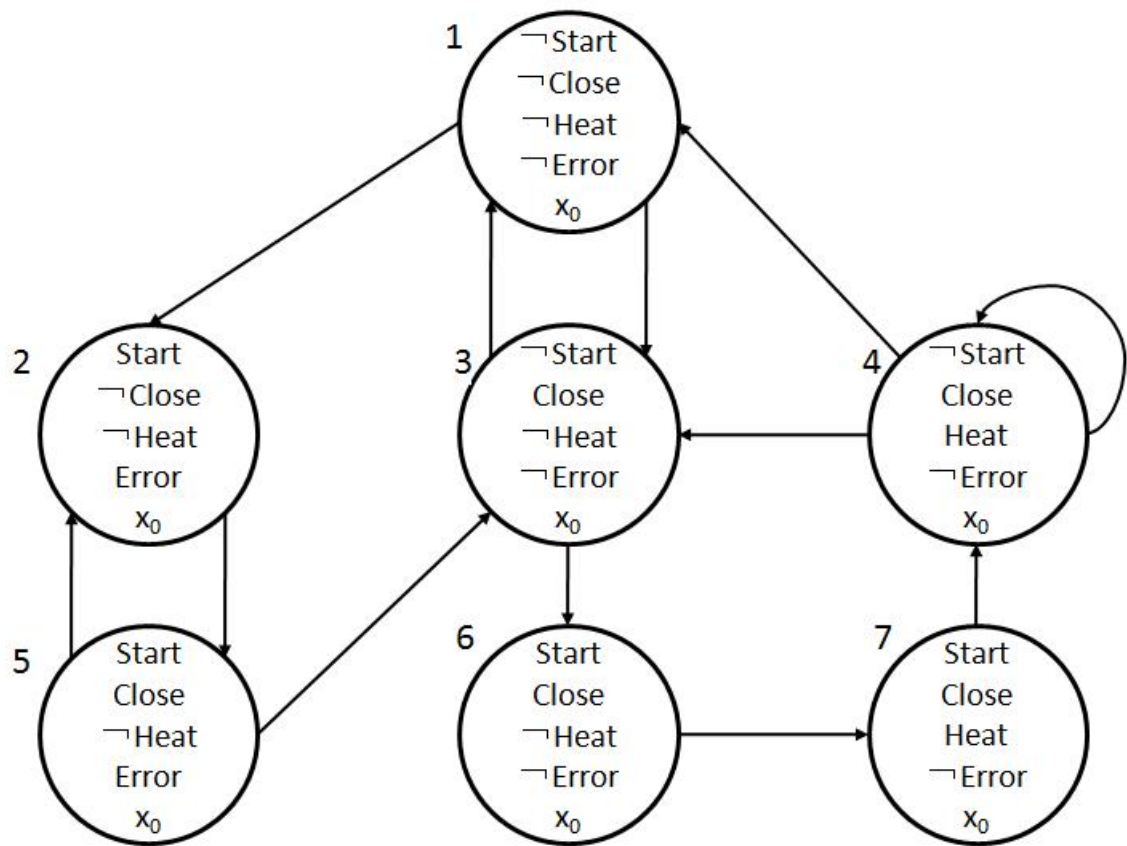


図 3.4:  $x_0$  の更新をしたクリプキ構造

## 第4章 関連研究

### 4.1 モデル検査器

本研究に使用した NuSMV 以外のモデル検査器について紹介する。

#### 4.1.1 SPIN

SPIN(Simple Promela Interpreter)[13][6] は 1980 年代に米国のベル研究所にて開発が開始された検証ツールであり，LTL で表された仕様を検証できるモデル検査器である．SMV とともに一般的に広まっているモデル検査器として有名である．日本での研究も盛んで，日本語で書かれた書籍も多い．

特徴としては，調べる状態の爆発を防ぐために，半順序法や状態データ圧縮などが施されている．

#### 4.1.2 UPPAAL

UPPAAL[14] は 1995 年にスウェーデンの Uppsala 大学とデンマークの Aalborg 大学によって開発された，リアルタイムシステムのモデリング，シミュレーションおよび検証を行うための総合開発環境である．UPPAAL では，グラフィカルなエディタを使い，システムを構成する各プロセスを時間オートマトンで記述する．また，CTL に時間の概念を導入して拡張した時相論理である時間計算木論理 (Time CTL, TCTL) [18] で表された仕様を検査できることも特徴である．

#### 4.1.3 LTSA

LTSA(Labelled Transition System Analyser)[9] は 1990 年代後半に英国のロンドン大学インペリアル校で開発されたモデル検査ツールであり，並行システムを検証対象としているモデル検査器である．並行システムの検証に必要なデッドロックやライブロックの検証をはじめ，LTL で表された仕様も検証できる．モデルの記述にローカル変数しか使えないなど，各種の制限はあるものの，他のモデル検査器に比べて使いやすいものとなっている．

## 4.2 応用研究

モデル検査のソフトウェアへの適用を行った研究として、コンパイラの最適化を対象としたものがある。この章ではこの研究について、いくつか紹介をする。

### 4.2.1 Lacey らの研究

Lacey らの研究 [7][8] は、時相論理 CTL に対して過去の時制を扱う演算子を加え、自由変数を導入して拡張した時相論理 CTL-FV を提唱した。この理論をもとに、モデル検査の手法を使って実際に最適化器を作成していく研究として、山岡 [16]、番 [1]、方 [19] などがある。特に、山岡の研究はモデル検査器として、今回使用した NuSMV と同種の SMV を使用している。

### 4.2.2 佐原の研究

佐原の研究 [17] は、コンパイラの最適化の正しさ、正確には最適化によるプログラムの意味の保存を検証するために、時相論理 CTL-FV によって各最適化の仕様を表し、最適化をかけるプログラムに対してモデル検査を行うというものである。

## 第5章 まとめ

今回実装した CTL\* のモデル検査器は、様々な最適化を施した既存のモデル検査器 NuSMV の、複雑な中身を壊さぬよう安全に実装を行うという考えから、NuSMV を外側から使用して検査するシステムをとっている。しかし、この場合ではモデルの各状態に対するラベル付けなどが、NuSMV の中と外で余計に行われることになってしまう。また、本システムでは入力にモデルのクリプキ構造の情報が必要であるが、これは他の入力である smv\* ファイルの情報から解析可能である。ただ、この解析部を本システムに組み込んだとしても、前述のとおり余計に解析を行うことになる。そこで、今後は NuSMV の拡張をする形で実装を行うことにより、より実用的な CTL\* のモデル検査器が出来上がるであろう。

# 謝辞

本研究を進めるにあたり多大なる御指導ご鞭撻を頂いた、東京工業大学 数理・計算科学専攻教授の佐々政孝先生に深く感謝の意を表します。

また、佐々研究室の皆様にはさまざまな面で助力を頂きました。あらためまして、ここに深くお礼申し上げます。

## 参考文献

- [1] 番伸宏, 胡振江, 箕一彦, 武市正人. Java プログラム最適化の宣言的記述とその効率的な実装. 第6回プログラミングおよびプログラミング言語ワークショップ (PPL2004), pp. 65–75, 2004.
- [2] Beatrice Berard, Michel Bidoit, Alain Finkel, Francois Laroussinie, Antoine Petit, Laure Petrucci, Philippe Schnoevenel, and Pierre McKenzie. *Systems and Software Verification*. Springer, 2001.
- [3] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, Vol. 8, No. 2, pp. 244–263, 1986.
- [4] E. M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.
- [5] Michael Huth and Mark Ryan. *Logic in Computer Science Modelling and Reasoning About Systems*. Cambridge University Press, 2004.
- [6] Gerard J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2004.
- [7] David Lacey, Neil D. Jones, Eric Van Wyk, and Carl Christian Frederiksen. Proving correctness of compiler optimizations by temporal logic. In *POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 283–294. ACM Press, 2002.
- [8] David Lacey, Neil D. Jones, Eric Van Wyk, and Carl Christian Frederiksen. Compiler optimization correctness by temporal logic. *Higher Order Symbol. Comput.*, Vol. 17, No. 3, pp. 173–206, 2004.
- [9] Ltsa homepage. <http://www.doc.ic.ac.uk/ltsa/>.
- [10] M. Reynolds. An axiomatization of full computation tree logic. *J. Symb. Log.*, No. 3, pp. 1011–1057, 2001.
- [11] Nusmv homepage. <http://nusmv.irst.itc.it/>.
- [12] Amir Pnueli. The temporal logic of programs. In *Proceeding of the 18th IEEE Symposium on Foundations of Computer Science*, pp. 46–77, 1977.
- [13] Spin homepage. <http://spinroot.com/spin/whatispin.html>.

- [14] Uppaal homepage. <http://www.uppaal.com/>.
- [15] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proceedings of the VIII Banff Higher order workshop conference on Logics for concurrency : structure versus automata*, pp. 238–266, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.
- [16] 山岡裕司, 胡振江, 武市正人, 小川瑞史. モデル検査技術を利用したプログラム解析器の生成ツール. 情報処理学会論文誌, Vol. 44, No. SIG13(PRO18), pp. 25–37, 2003.
- [17] 佐原聡一郎, 佐々政孝. 時相論理を用いたコンパイラ最適化器の実行の正しさの検査. コンピュータソフトウェア, Vol. 25, No. 1, 2008.
- [18] 磯部祥尚, 櫻庭健年, 田口研治, 田原康之, 糸野文洋, 田中謙. ソフトウェア科学基礎 最先端のソフトウェア開発に求められる数理的基礎. 近代科学社, 2008.
- [19] 方玲, 佐々政孝. 双方向 CTL による Java 最適化器の生成. 情報処理学会論文誌, Vol. 48,