卒 業 論 文

UPPAALを用いた自動運転車の群制御アルゴリズムのモデル化と検証

Modeling and verification
of autonomous-vehicle group control algorithms
using UPPAAL

指導教員　中村 正樹 准教授

富山県立大学工学部 電子・情報工学科

学籍番号：1515024

氏名　佐原 優衣

提出年月　2019年2月

目 次

第1章　はじめに

1.1　背景

自動運転技術は発達し続けている。自動運転は，搭載される技術によってレベル1からレベル5までに分けられており，現在，日本国内では，運転者支援を主としたレベル2までが市販車に採用されている。今後，高速道路や，限定地域での特定条件下での完全自動運転を行うレベル4の車両の普及が目指されている。

完全自動運転の普及の環境の一例として，アラブ首長国連邦において再生可能エネルギーを利用し，二酸化炭素を排出しないゼロカーボンを目指すマスダールシティプロジェクトが2006年に始まった。このプロジェクトでは道路交通は自動運転車のみで構成される予定である。住民が任意の時刻に自動運転車に乗降し都市空間内を移動することを想定しているため，大量の車両の配備が必要となる。道路上の車両密度が高くなるため，渋滞やデッドロックが発生することが想定される。したがって，個々の車両だけではなく，自動運転車群が効率的に走行するアルゴリズムが必要となる。

しかし，効率的に制御された自動運転車が安全かどうかアルゴリズムから一目では判断しきれない。そこで本研究では群制御アルゴリズムが欲しい性質を持っているかどうかを検証する手法を提案する。

図 1.1: マスダール・シティの完成イメージ [1]

1.2 目的

自動運転車の群制御アルゴリズムを形式的に記述し，モデル検査を用いて，性質を検証する。

1.3 論文の構成

本論文は，2 章で本研究で用いるモデル検査の概説と使用するモデル検査ツールの概説，3 章で単一の交差点における車両のモデル化と検証，4 章で検証にかかる時間や検証の質，5 章でまとめを述べるといった構成である。

[1] 出典：Masdar 社

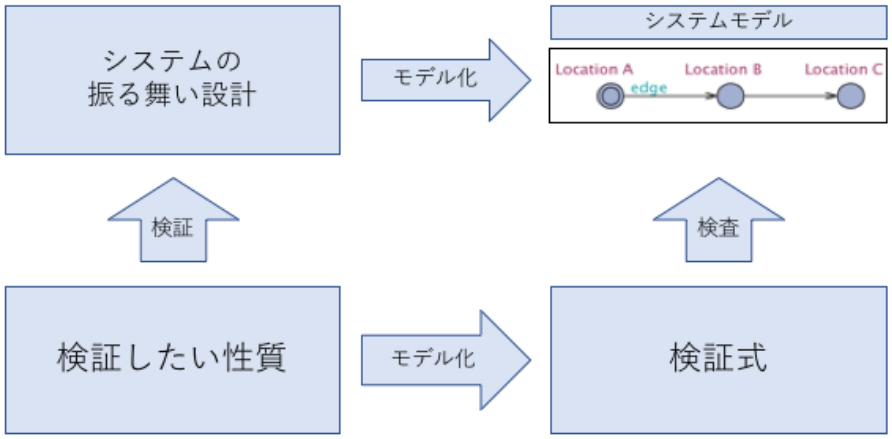
システムの
振る舞い設計
モデル化
システムモデル
Location A
Location B
Location C
edge
検証
検査
検証したい性質
モデル化
検証式

図 1.2: モデル検査とは

第2章　モデル検査

本章では文献 [1] からモデル検査と UPPAAL の概説を行う。

2.1　モデル検査

モデル検査は，システム上で起こり得る状態を網羅的に調べることにより設計の誤りを発見する自動検証手法の一種である。モデル検査手法は，システムの振る舞いの設計，および検証したい性質をそれぞれモデル化し，ツールを用いて，システムが性質を満たしているかを調べる。

モデル検査において，システムの動作を表現するシステムモデルを作成する必要がある。ソフトウェア開発のどの段階でモデル検査を活用したいか，もしくは，何をどの程度検証したいかによって，どのような情報をもとにどのようにシステムモデルを作成するかが変わってくる。専用のシステムモデルを入力とするモデル検査を設計モデル検査，ソースプログラムを入力とするモデル検査をプログラムモデル検査と呼ぶ。これらのモデル検査がソフトウェア開発の流れの中での活用例を図 2.1 に示す。図 2.1 にソフトウェアの品質向上のために行われる手順も挙げた。設計モデル検査は設計レビューを，プログラムモデル検査はコードレビューをそれぞれ補完する位置付けである。

2.1.1　様々なモデル検査ツール

次に，いくつかのモデル検査ツールを特徴と共に例示する。代表的なモデル検査ツールには，処理が高速で大規模なモデルを扱える NuSMV，並列処理やマルチスレッドの設計モデルを扱える SPIN，GUI ベースの入力による時間制約を扱える UPPAAL などがある。

NuSMV は状態遷移図からのモデル化に向いており，また，各状態が満たす論理式などを用いて記号的に検査を行う，莫大な状態数を持つ系に対して検査が可能なシンボリックモデル検査ツールである。

SPIN は Promela という専用の言語を用いて検証対象をモデル化する。動作主体を複数定義し，各動作主体の動作は C 言語に似た文法で記述される。複数の動作主体の状態の組み合わせの数は，例え一動作主体あたりの状態数が比較的少なくとも，非常に大きくなることが起こりがちである。これを状態爆発と称される。SPIN は状態爆発を緩和するために，半順序簡約，ビットステートハッシュ，状態ベクトル圧縮，データフロー・制御フロー解析によるスライシングなどの諸技術が用いられている。

SPIN は Promela という専用の言語を用いて検査対象をモデル化する。SPIN の最大の特徴はモデル検査そのものを実施せず，対象そ固有の C 言語ソースを生成する。メモリ使用量を削減し，性能向上と共に，モデルに使用者が自由に C 言語コードを追加できる利点がある。

UPPAAL の最大の特徴である時間が扱えることの利点を記述する。イベントの発生時刻や処理時間，これらの時間的なズレの三点について任意に設定できるため，時間が扱えないモデル検査と違い，応答時間などの時間制約を検証対象にすることが可能である。また，イベントの発生と特定の処理の開始を簡単に記述できる。

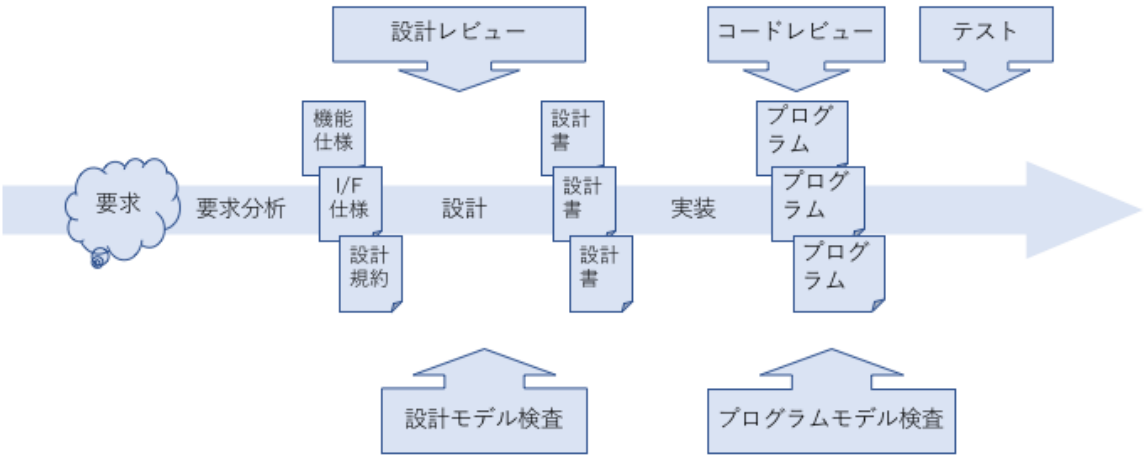

図 2.1: ソフトウェア開発プロセス

2.2 モデル検査ツール UPPAAL

本節は，UPPAAL について概説する。

UPPAAL はシステムモデルの入力を GUI ベースにより定義している。このため，作成したシステムモデルが直感的に把握しやすい。入力したシステムモデルに対して，GUI ベースでシミュレーション実行とステップ実行が可能である。シミュレーション画面では，各プロセスの現在状態と変数の値，状態遷移図とメッセージシーケンスが表示される。

検証は検証したい性質を検証式で入力する。検証はすべての可能性のある実行パスに対して網羅的に検査を行う。検証結果は，入力した検証式に対して成否が

緑か赤で示される。性質に反した場合は，反するまでの実行履歴が反例として示される。反例の表示はシミュレーション画面で行われ，ステップ単位でトレースすることで，各プロセスの状態や変数値の変化を確認可能である。また時間制約を含む検証に関して,「最短時間で違反状態に到達する反例の出力」という機能を持つ。通常出力されるのは任意の一種類ではあるが，特に初期状態から検証したい性質に反するまでの経過時間が最短となる反例を出力する機能である。検証したい性質として,「仕事が完了することがない」という条件を与えることにより，仕事が完了する手順が反例となるが，仕事が完了する手順の中で最短時間のものを出力することになる。

2.2.1 UPPAALの操作

本節では，例題を用いてUPPAALを使用する。

例題の説明：4台の車が出発地点から到着地点まで，それぞれ違う方向から一つの交差点に進入し，通過していくとき4台の車が全て通過し終わるのにかかる最小時間を検証する。なお，この交差点には右折レーン，信号がないものとする。北から南に向かう車ns，南から北にぬける車sn，東から西にぬけるew，西から東にぬけるwe，の4台は時速約30kmほどで走っており，南北同士，東西同士の場合はお互いに止まることなく通過するが，nsとewのような交差する場合は先に交差点への進入権を獲得した方が先に通過することとなる。そのため交差点への進入権をいつ獲得し，交差点の通過にかかる時間も考慮する。

システムモデルの入力：UPPAALはシステムモデルをGUI言語で作成する。UPPAALでは本例題における，車一台一台をエージェントと呼び，一台の車のおける出発地点から目的地まで通過するという一連の動作のことをプロセスと呼ぶ。同一の挙動を示すプロセスの定義を個別に行わなくてもすむように，テンプレートという概念を用いている。そのため，システムモデルの作成では，まずプロセスの挙動を定義したテンプレートの定義から行う。図2.2に示すように状態遷移図をベースとしたおり，　シミュレーションと検証：

時間表現：

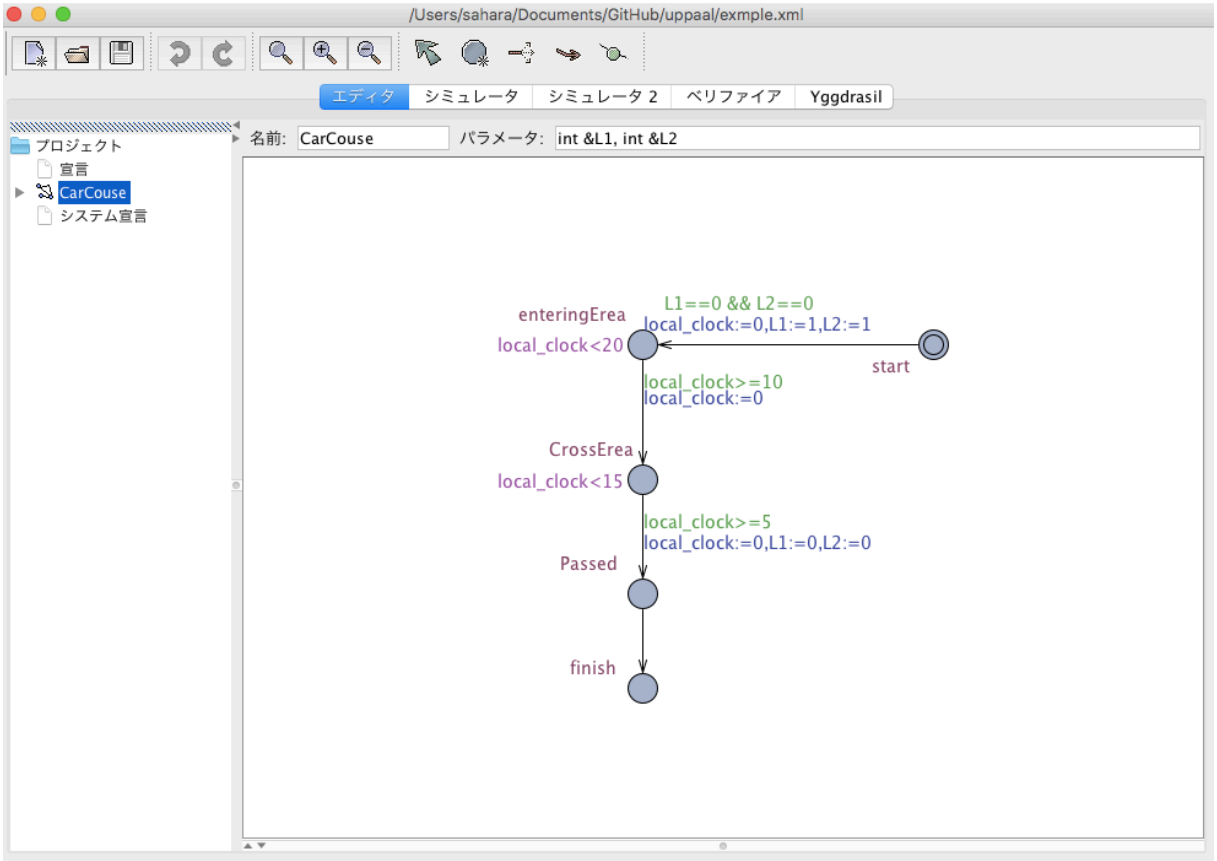

図 2.2: システムモデルの入力部分

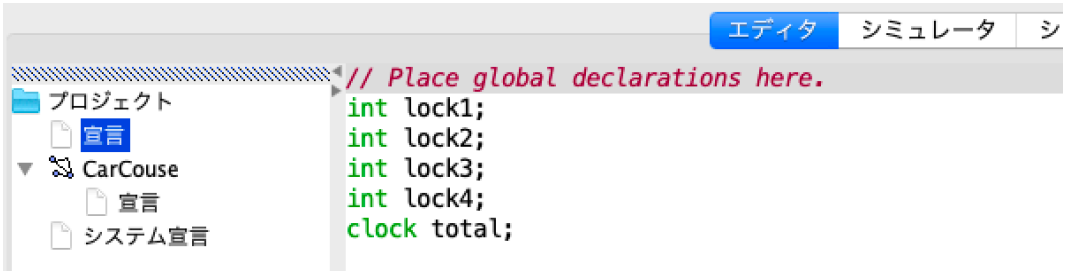

図 2.3: 大域情報編集ペイン

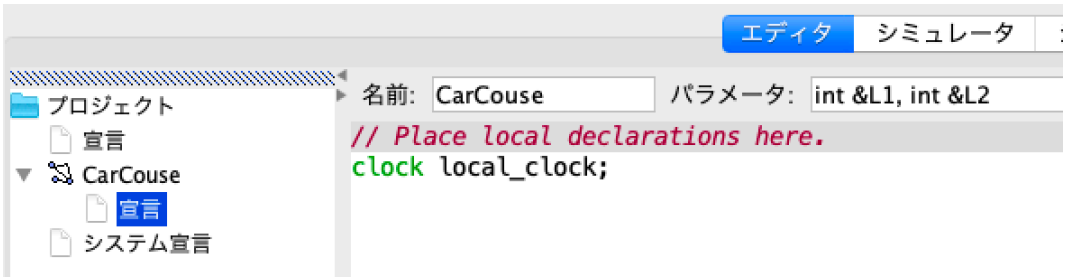

図 2.4: 局所的情報編集ペイン

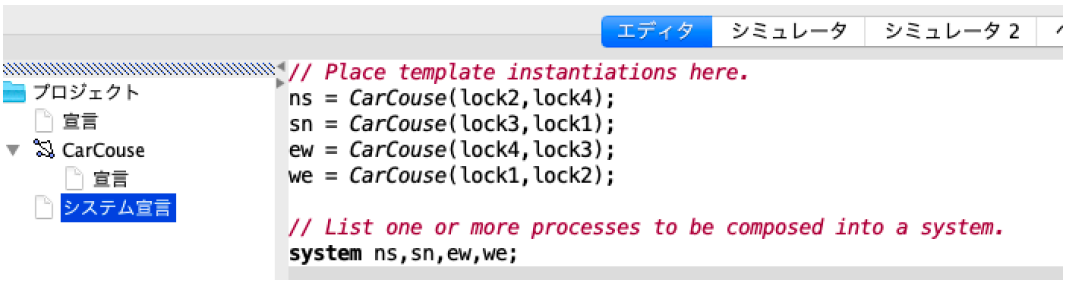

図 2.5: システム定義編集ペイン

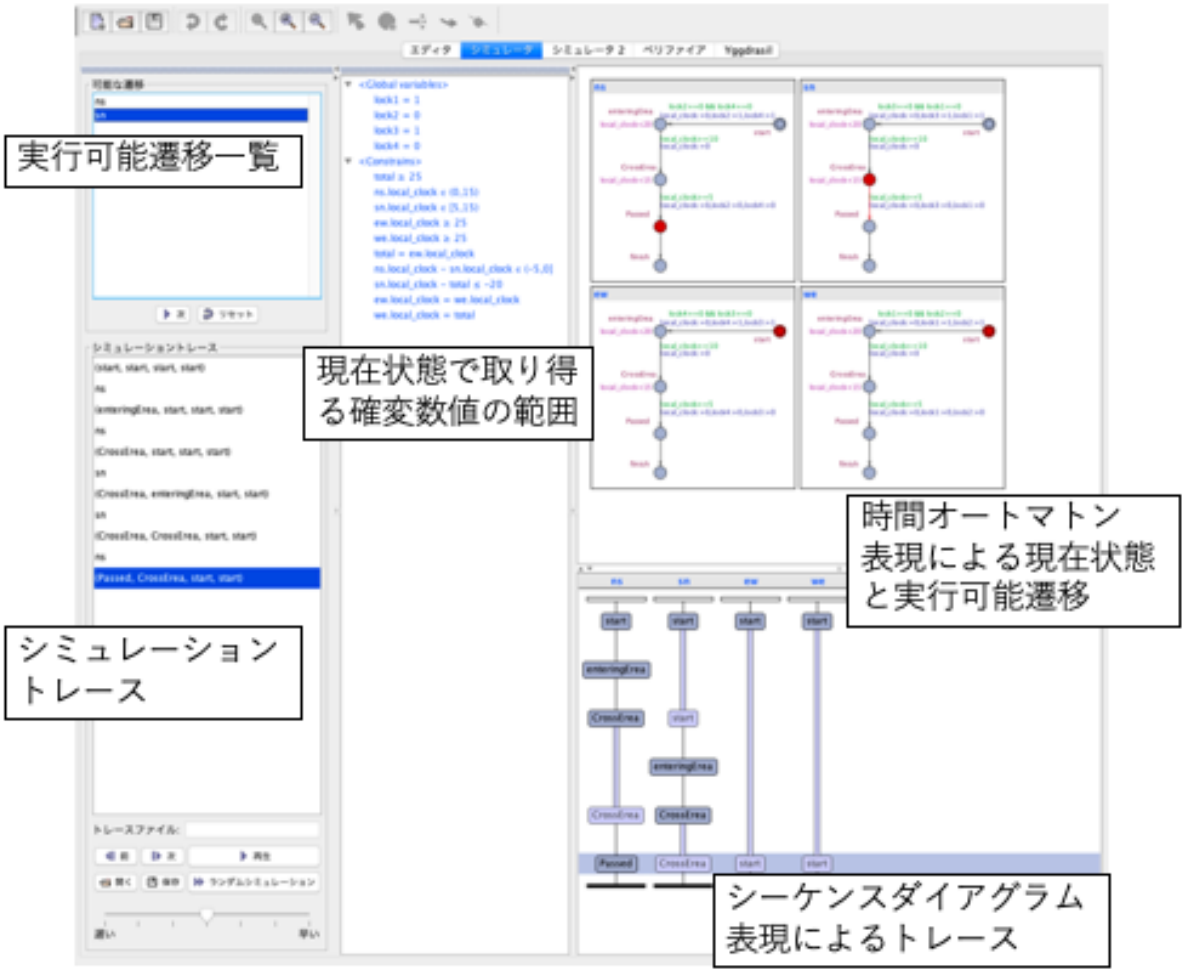

図 2.6: シミュレーション機能


エディタ
シミュレータ
シミュレータ 2
ベリファイア
Yggdrasil
概要
E<> (sn.CrossErea && ns.CrossErea)
A[] not(sn.CrossErea && ew.CrossErea)
A[] not deadlock
検査
追加
削除
注釈表示
検証式一覧表示
クエリ
E<> (sn.CrossErea && ns.CrossErea)
検証式編集エリア
注釈
コメント編集エリア
状況
ローカルサーバへの直接接続が確立されました
(Academic) UPPAAL version 4.1.19 (rev. 5648), October 2014 -- server.
A[] not deadlock
Verification/kernel/elapsed time used: 0.004s / 0.003s / 0.006s.
Resident/virtual memory usage peaks: 5,472KB / 4,340,352KB.
属性は満たされませんでした
A[] not(sn.CrossErea && ew.CrossErea)
Verification/kernel/elapsed time used: 0.003s / 0.003s / 0.006s.
Resident/virtual memory usage peaks: 5,516KB / 4,340,352KB.
属性は満たされました
A[] not(sn.CrossErea && ew.CrossErea)
Verification/kernel/elapsed time used: 0.001s / 0s / 0s.
Resident/virtual memory usage peaks: 5,524KB / 4,340,352KB.
属性は満たされました
E<> (sn.CrossErea && ns.CrossErea)
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 5,524KB / 4,340,352KB.
属性は満たされました
検証状況表示エリア

図 2.7: モデル検査機能

第3章　群制御アルゴリズムのモデル化と検証

3.1　他車の動作を視野に入れた交差点の通過モデル

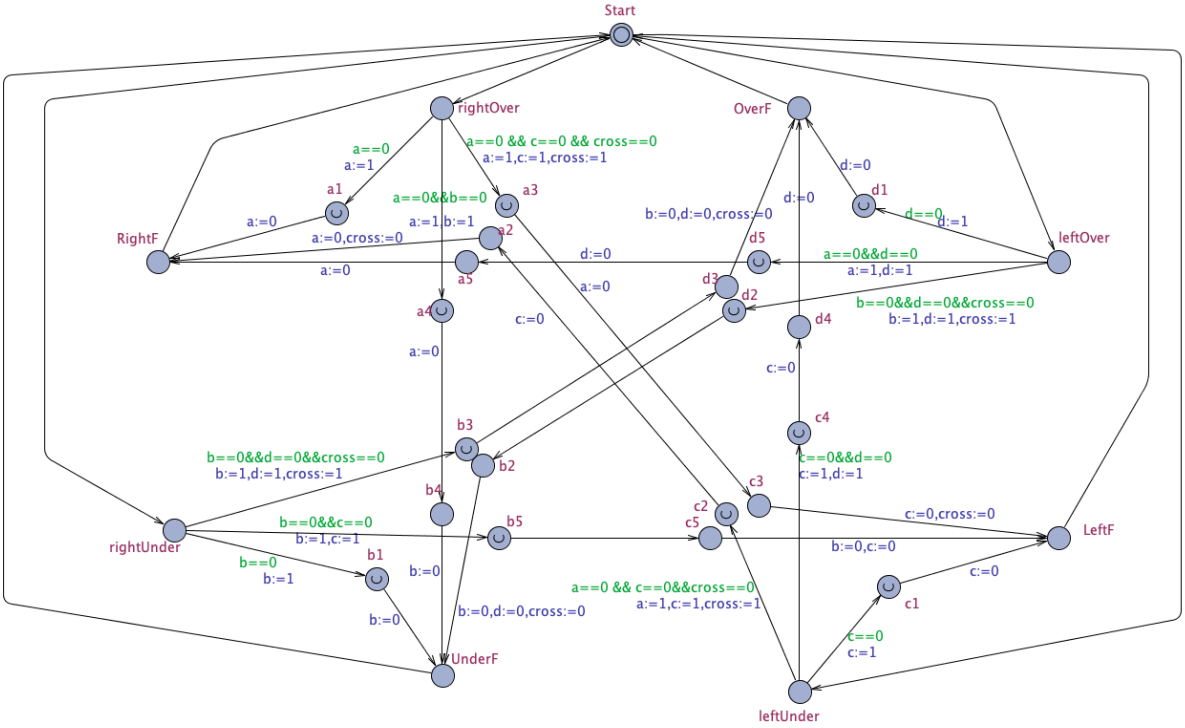

図 3.1: 他車の行動を視野に入れた交差点のモデル

```
A[] not deadlock
Verification/kernel/ellapsed time used: 0s / 0s / 0.001s.
Resident/virtual memory usage peaks: 5,776KB / 4,354,688KB.
属性は満たされました
```

図 3.2: 他車の行動を視野に入れた交差点のモデルを 1 台の車が動作した時の検証結果

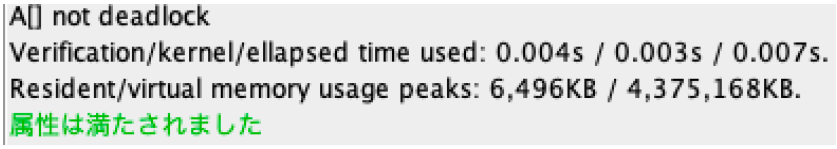

図 3.3: 他車の行動を視野に入れた交差点のモデルを 2 台の車が動作した時の検証結果

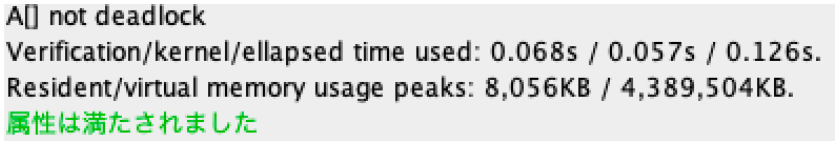

図 3.4: 他車の行動を視野に入れた交差点のモデルを 3 台の車が動作した時の検証結果

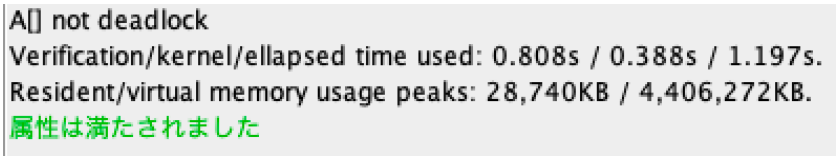

図 3.5: 他車の行動を視野に入れた交差点のモデルを 4 台の車が動作した時の検証結果

図 3.6: 他車の行動を視野に入れた交差点のモデルを 5 台の車が動作した時の検証結果

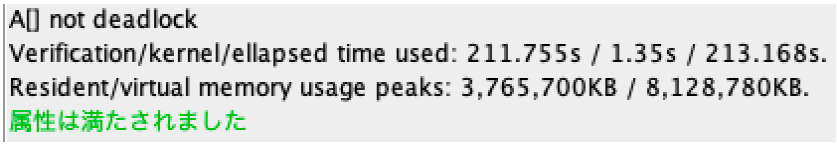

図 3.7: 他車の行動を視野に入れた交差点のモデルを 6 台の車が動作した時の検証結果

3.2　交差点の使用権を獲得するモデル

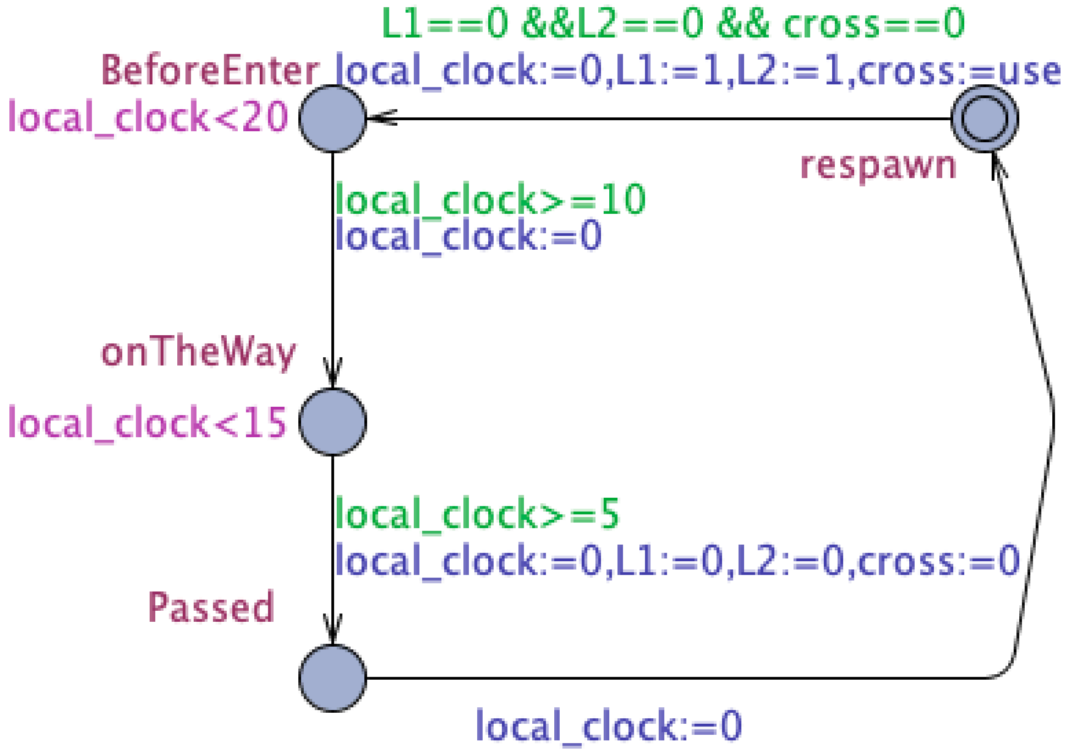

図 3.8: 交差点の進入時に使用権を獲得するモデル

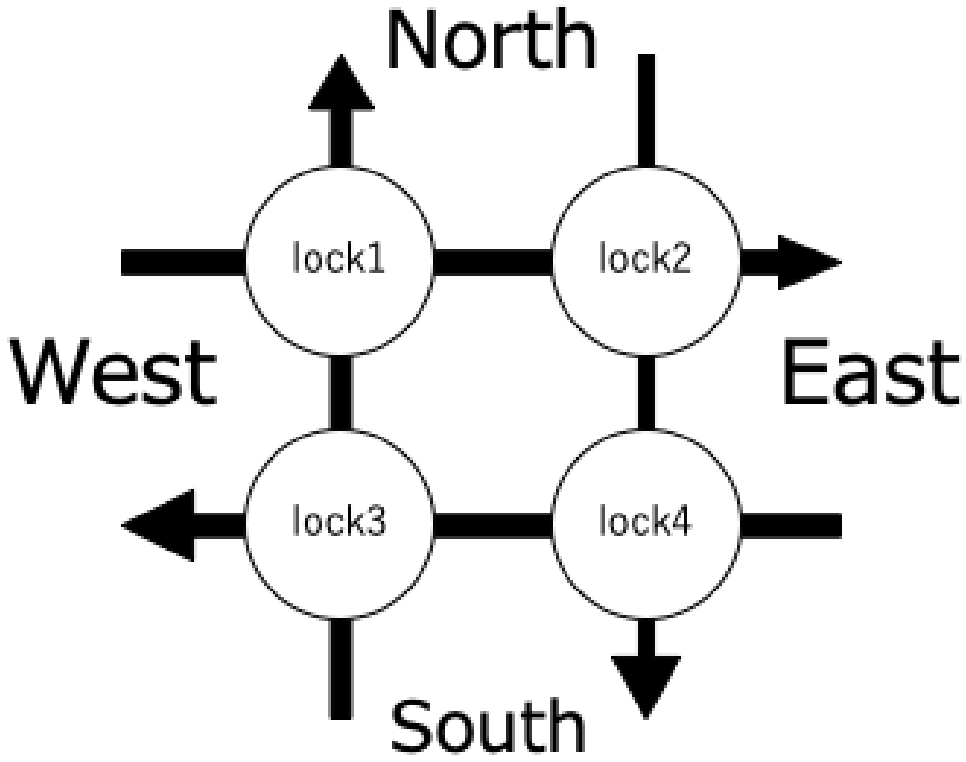

図 3.9: 交差点の進入権

第4章　考察

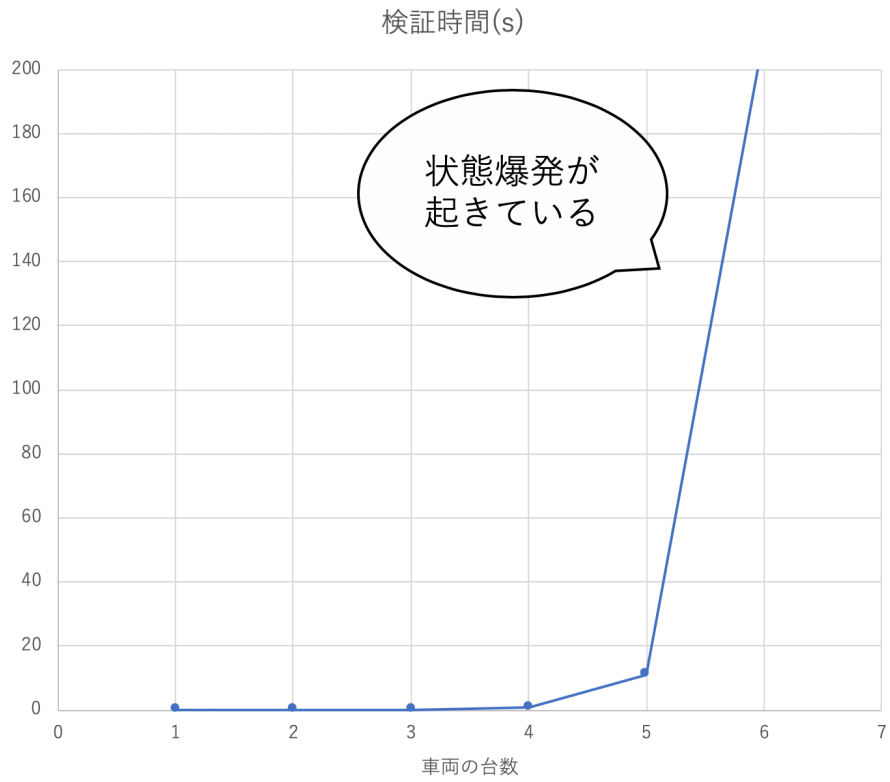

図 4.1: 車両の台数と検証にかかる時間の関係

第5章　おわりに

謝 辞

参 考 文 献

[1] 長谷川哲夫，田原康之，磯部祥尚，UPPAAL による性能モデル検証ーリアルタイムシステムのモデル化と検証ー,（株）近代科学社，2012