UPPAAL を用いた自動運転車の群制御アルゴリズムの性能モデル検証

富山県立大学 佐原 優衣 中村 正樹 榊原 一紀 玉置 久

1 はじめに

近年，自動運転技術が急速に発達している。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。マスダールはアラブ首長国連邦の一つアブダビ首長国の首都アブダビの近郊で図 1 の様な人口約 5 万人，面積約 6.5km^2 の人工都市として計画されている。

図 1 マスダール・シティの完成イメージ [*2]

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。

本研究では群制御アルゴリズムが安全性に関わる衝突回避やデッドロック回避，効率性に関わる時間制約などの性質を満たすかどうかを検証する手法を提案する。

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。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。本研究では，時間オートマトンによる時間制約検証が行えるモデル検査ツール UPPAAL[6] を採用する。

2 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??に示す。図??にはソフトウェアの品質向上のために行われる手順である設計レビュー，コードレビュー，およびテストも挙げた。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。

3 群制御アルゴリズムのモデル化と検証

UPPAAL を用いて交差点を通過する 1 台の自動運転車の挙動をモデル化する。交差点は 2 車線対面通行で右折用レーンはなく，信号もない交差点とする。3.1 節では時間は扱わず，任意の方向へ進む車両をモデル化する。3.2 節では時間に関する条件を用いることで，交差点通過時間を記述する。3.3 節では，全ての車両が交差点を通過するのに掛かる最小時間を検証する。

4 時間制約のない進行方向を固定しない交差点

本節では，経路選択をして交差点を通過する車両のモデルを作成する。信号のない交差点を無秩序に通過すると衝突

[*2] 出典：Masdar 社

などが起きる可能性がある。したがって今回は交差点に使用権というものを設定し，交差点を通過するにはこの使用権が取得できた車両が通過できることとする。

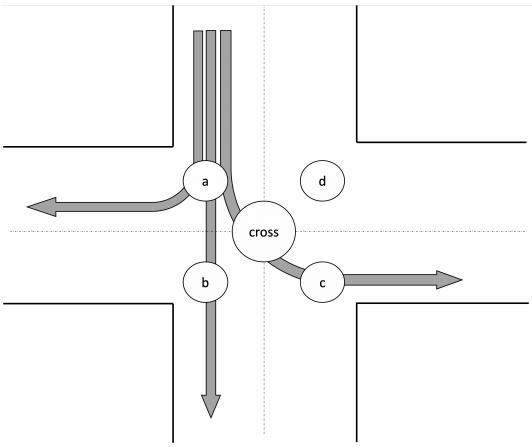

図 2　交差点における使用権の鍵の組み合わせモデル

交差点に 5 つの鍵を設定し，その組み合わせで交差点を通過可能とする。図 2 は 5 つの鍵を用いた交差点の車両の進行モデルである。図上からの進入に対して，直進，右折，左折の選択肢があり，直進は鍵 a,b を，右折は鍵 a のみを，左折は鍵 a,c,cross をそれぞれ取得する。該当の鍵を取得可能な時，その進行方向へ進入可能となる。図 2 を基にして，1 台の車両の交差点通過を表すオートマトンを作成する。

4.1　時間オートマトンの作成

図 3 は初期状態 Start から交差点に進入し，通過後 Start に戻ることで，上下左右の全ての方向から直進，右折，左折を非決定的に選択することを繰り返す 1 台の車両のオートマトンである。Start から交差点進入前状態 rightOver に遷移したとき，車両は，直進，左折，右折の 3 つの選択肢がある。直進するときは，交差点通過中状態 a4 への遷移可能条件として大域変数 a==0 かつ,b==0 すなわち使用されていないとき，遷移時に a=1 と b=1 として使用権を取得し，b4 への遷移時に a=0 として使用権の一部を解放する。そして，b4 から交差点通過後状態 UnderF への遷移時に b=0 としてもう 1 つの使用権も解放する。右折するときは，通過中状態 a1 への遷移可能条件を a==0 とし，遷移時に a=0 と更新し，使用権を取得する。通過後状態 RightF への遷移時に a=0 として使用権を解放し Start に戻る。左折するときは，通過中状態 a3 への遷移条件を a==0 かつ，c==0 かつ，cross==0 とし，遷移時に 3 変数を 1 に更新して使用権を取得する。通過中状態 c3 への遷移時に a=0 として鍵 a を解放し，c3 から通過後状態 LeftF への遷移時に使用権を解放し，Start に戻る。以上の遷移を交差点に進入する 4 方向についてすべて記述する。

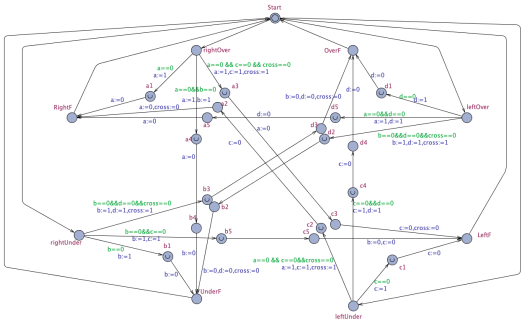

図 3　進行方向を固定しない交差点の時間オートマトン

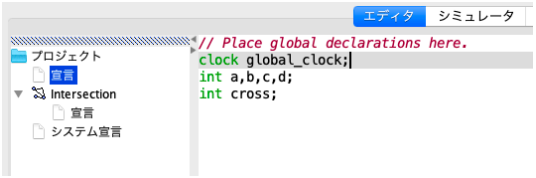

図 4　時間制約のない交差点の時間オートマトンの大域情報

4.2　シミュレーション

前節で作成した時間オートマトンのインスタンスを 2 つ宣言し，シミュレーションを行う。図 5 では，Car1 は leftOver から右折して交差点を通過中状態で，Car2 は rightOver で交差点進入前状態である。このとき大域変数は図 6 となっている。したがって，Car2 は直進右左折それも選択可能となっている (図 7)。

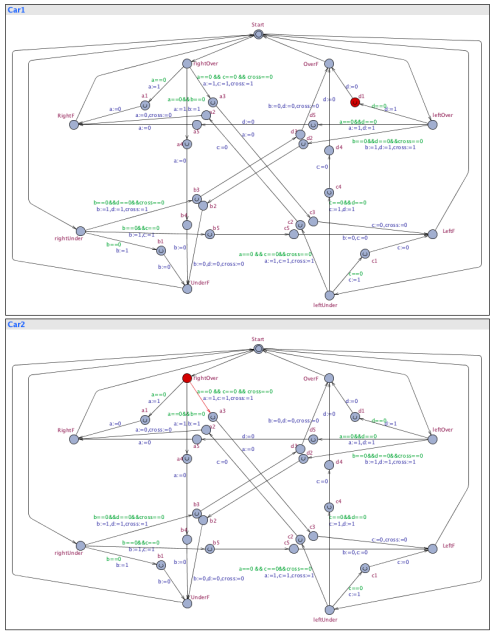

図 5　時間制約のない交差点の時間オートマトン

4.3　モデル検査

UPPAAL では，どれだけ時間が経過しても，いずれのプロセスも遷移できないことをデッドロックという。本モデルでデッドロックが起きないか検証する。本例題におけるデッドロックは同じ鍵を同時に使うことや，Start に戻って

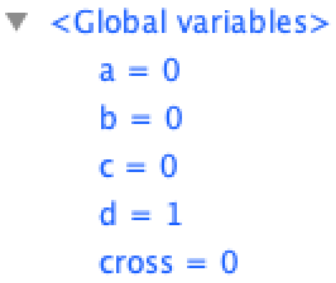

図 6　時間制約のない交差点の時間オートマトンの大域変数値

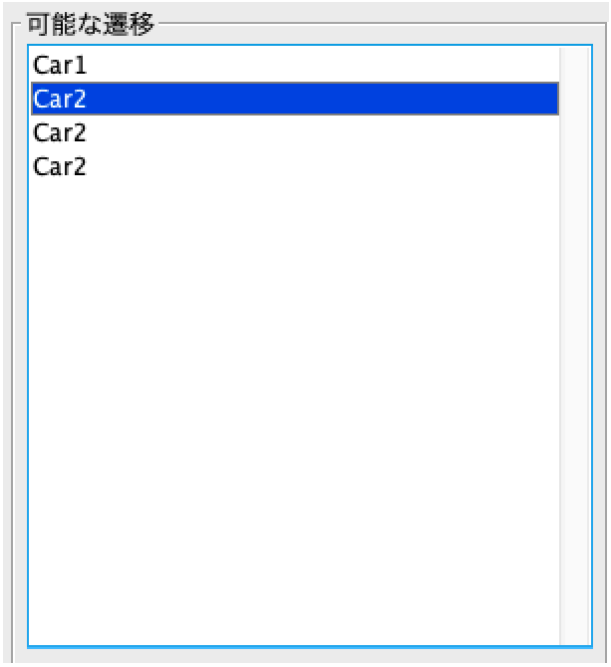

図 7　時間制約のない交差点の時間オートマトンの実行可能遷移一覧

これないことを指す。すべての実行列で常にデッドロックにならないことを表す検証式は次のように記述される：

```
A[] not deadlock
```

検証結果は図 8 のように示される。

プロセスの車両インスタンスを 1 から 6 まで 1 ずつ増やしながらデッドロック検証を行った。表 1 は車両の台数に対する検証にかかった時間である。図 9 はそれをグラフ化したものである。

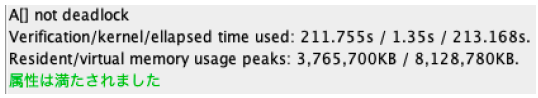

図 8　進行方向を固定しない交差点のデッドロック検証結果

5　時間制約を用いる進行方向を固定する交差点

本節では，交差点に進入する車両の進行方向を固定して，時間オートマトン [8] を作成する。車両 1 台の挙動は，交差点進入前，交差点通過中，交差点通過後の 3 つの状態で記述

表 1　車両の台数とデッドロック検証にかかる時間

車両の台数	検証時間 (s)
1	0.00
2	0.004
3	0.068
4	0.808
5	10.873
6	211.755

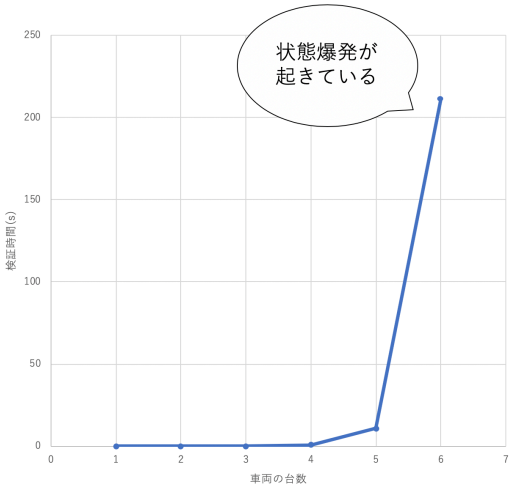

図 9　車両の台数と検証にかかる時間の関係

する。交差点進入前に交差点の使用権を取得し，通過後に使用権を解放する。遷移可能条件と状態不変条件に時間に関する条件を与えることによって，通過にかかる時間や使用権を何秒前に取得しなければならないかを記述できる。

5.1　一方通行の 2 車線で構成される交差点モデル

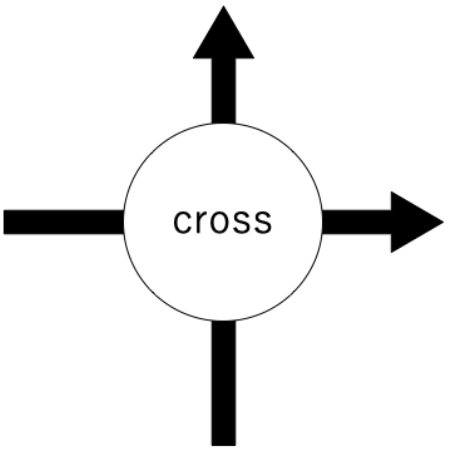

図 10　交差点の使用権

一方通行の交差点で互いの進行方向が交わる時，交差点を図 10 のようにモデル化する。cross は交差点の使用権を表現しており，使用権を取得できた車両が交差点を通過するモデルである。

5.1.1 時間オートンマトンの作成

交差点に直進する車両 2 台の進行方向が互いに交差するとき，衝突回避するためには交差点に同時に進入するのは 1 台までにする。交差点進入時に使用権を取得する車両の時間オートマトンを作成する (図 11)。respawn は車両の初期状態かつ，任意の時刻に車両が発生する状態である。BeforeEnter は車両の交差点進入前の状態である。crossArea は交差点通過中の状態である。Passed は交差点通過後の状態である。respawn から BeforeEnter への遷移可能条件の cross==0 は交差点の使用権が未獲得であることを示しており，遷移時に cross==1 と更新して交差点使用権を獲得する。同時にタイマーである local_clock を 0 にして，BeforeEnter の時間を測定する。BeforeEnter から crossArea の遷移でも同様にし，交差点手前の約 5 秒から 10 秒前までに使用権を獲得し，交差点通過に 2 秒から 5 秒弱かかるということが記述されている。

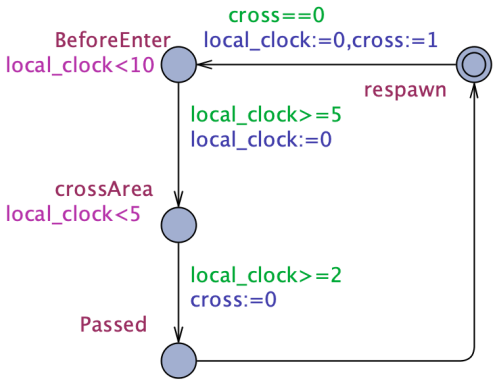

図 11　交差点進入時に使用権を取得する車両 1 台の時間オートマトン

5.1.2 シミュレーション

図 12 は，前節で定義したテンプレートから生成された 2 つのインスタンスが合成された時間オートマトンに対するシミュレーションのスクリーンショットである。左のプロセス vertical を縦方向に直進してるとし，右のプロセス horizon を横方向に直進しているとする。現在状態は vertical が交差点の使用権を獲得し，進入前状態で，horizon が交差点通過後の状態である。

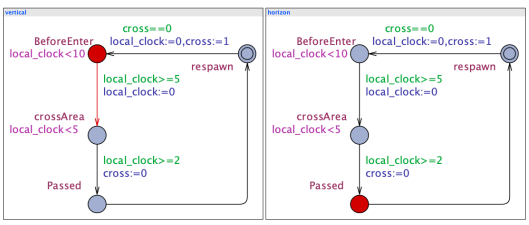

図 12　垂直に交差点に進入する車両の時間オートマトンの合成

5.1.3 モデル検査

このモデルでデッドロック検証を行った結果を図 13 に示す。

```
A[] not deadlock
Verification/kernel/ellapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 4,928KB / 4,317,824KB.
属性は満たされました
```

図 13　一方通行の 2 車線で構成される交差点のデッドロック検証結果

5.2 東西南北それぞれから交差点に直進する時間オートマトン

直進して交差点を通過する車両の進行方向に対して，他の車両の進行方向が垂直であったり，平行であったりする時の交差点の使用権の獲得方法を考える。前例では，cross が交差点の使用権そのものでそれの有無で進入を決定していたが，本例では，進行方向が平行となる場合は同時に進入可能としたい。したがって交差点の使用権を図 14 に示すように 4 つの鍵の組み合わせで管理したい。例えば，西から東へ進行する車両は lock1 と lock2 を取得する。北から南へ進行する車両は lock2 と lock4 を取得しようとするが，既に lock2 が取得されているためこの車両は交差点へ進入できない。一方で前者に対して，東から西へ進行する車両は lock3 と lock4 を取得できるため，交差点へ進入する。

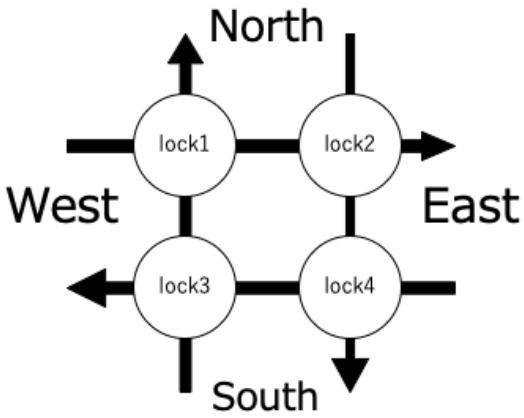

図 14　交差点の使用権を管理する 4 つの鍵の組み合わせ

5.2.1 時間オートマトンの作成

使用権を 4 つの鍵で管理する交差点に直進する車両の時間オートマトンを作成する (図 15)。遷移可能条件をパラメータ (L1,L2) で記述し，それぞれが取得する鍵に紐付けている。初期状態 respawn から交差点進入前状態 BeforeEnter へ遷移時にふたつの鍵 L1 と L2 を 1 に更新して使用権を取得する。前例と同様にこのオートマトンも直進のみのため，タイマーである local_clock は同内容を記述している。交差点通過中状態 crossArea から交差点通過後状態 Passed への遷移時に鍵の解除として L1==0 と L2==0 と記述した。

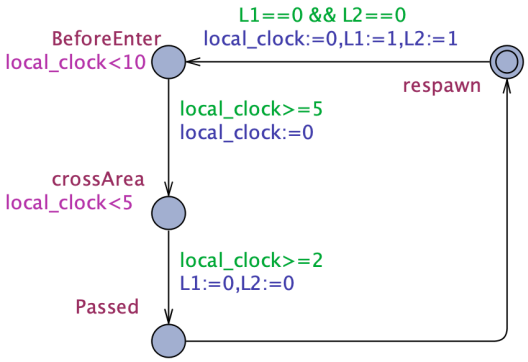

図 15　使用権を 4 つの鍵で管理する交差点に直進する車両の時間オートマトン

5.2.2　シミュレーション

北から南へ直進する車両を sn，南から北へ直進する車両を ns，東から西へ直進する車両を ew，西から東へ直進する車両を we として，インスタンスの宣言をした (図 17)。図 16 では，ew が交差点を交差点通過中で，lock3 と lock4 が取得されている。lock1 と lock3 を取得できた we が交差点進入前状態である。

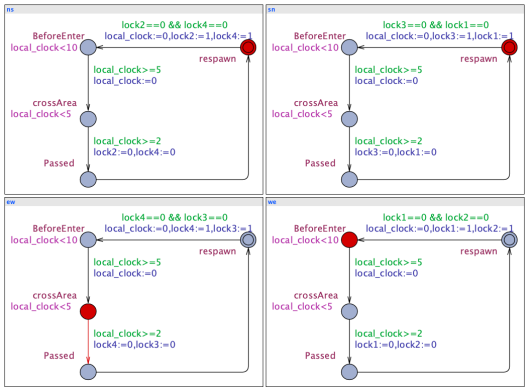

図 16　交差点に直進する車両の時間オートマトンの合成

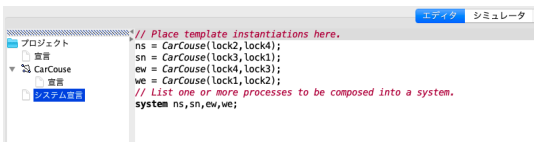

図 17　4 車両のシステム宣言

5.2.3　モデル検査

デッドロック検証を行った結果を図 18 に示す。

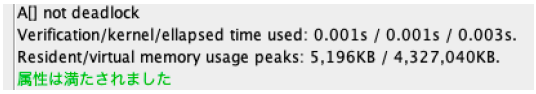

図 18　4 車線で構成される交差点のデッドロック検証結果

5.3　交差点を直進・左折・右折する車両の時間オートマトン

交差点を直進する車両と左折する車両と右折する車両を考慮したモデルを考える。前述までとは違い，右折する場合もあるため，4 つの鍵の組み合わせでは右折する車両同士での衝突が起こる可能性がある。したがって前述の 4 つの鍵に加えて，5 つ目の鍵を右折用の鍵として cross を用意したモデルを考える。このモデルで同時通行可能な組み合わせの 2 つの例を図 19 と図 20 に示す。

5.3.1　時間オートマトンの作成

このモデルから時間オートマトンを作成する (図 21)。鍵 cross は 3 つ目のパラメータ use の値と照らし合わせる。パラメータ use は右折用の鍵を使用するかどうかを車両の固定情報として保持している。初期状態 respawn から交差点進入前状態 BeforeEnter への遷移可能条件をパラメータ L1，L2，右折用鍵 cross が取得可能であることとした。また，左折時は，パラメータは 2 つあるが，取得する鍵は 1 つである。そのため，南から東へ右折する車両と，東から南へ左折する車両と，西から北へ左折する車両の 3 台が同時に交差点に進入可能となっている。

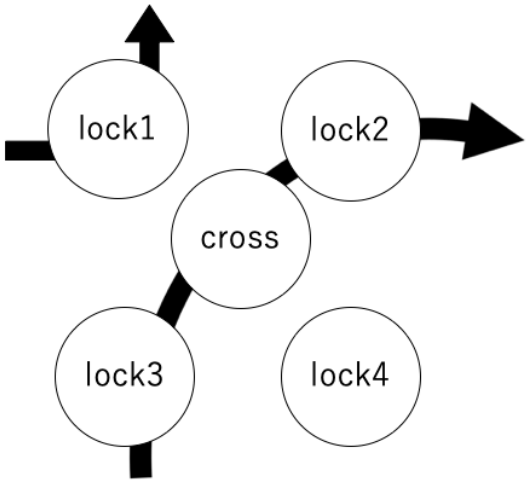

図 19　使用権の取得例：右折と左折

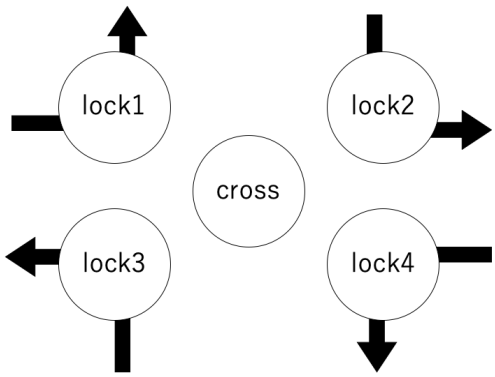

図 20　使用権の取得例：4 台の左折

5.3.2 シミュレーション

北から南へ直進する車両を sn，南から北へ直進する車両を ns，東から西へ直進する車両を ew，西から東へ直進する車両を we，北から東へ左折する車両を ne，南から西へ左折する車両を sw，東から南へ左折する車両を es，西から北へ左折する車両を wn，北から西へ右折する車両を sw，南から東へ右折する車両を se，東から北へ右折する車両を en，西から南へ右折する車両を ws として，インスタンスの宣言をする (図 23)。上記の 3 台の車両が交差点に進入する例が図 22 のように，右折する se が通過中状態 crossArea の時，左折する es も状態 crossArea で，もう一つの wn が状態 BeforeEnter となっている。

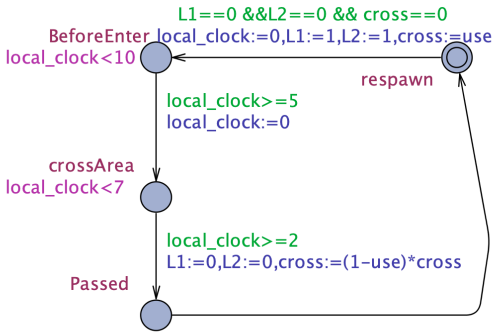

図 21　交差点を通過する車両の時間オートマトン

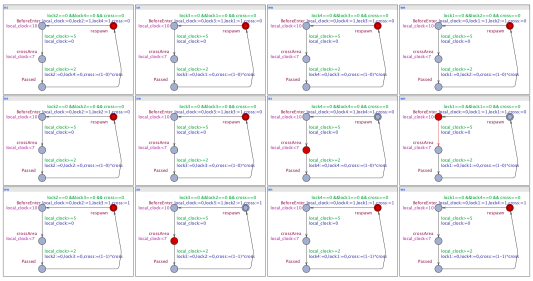

図 22　交差点を通過する車両の時間オートマトンの合成

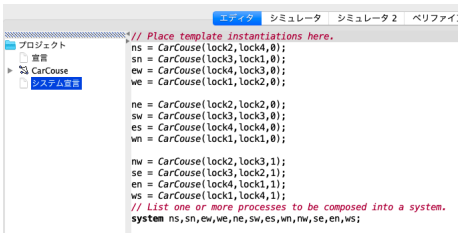

図 23　12 車両のインスタンスの宣言

5.3.3 モデル検査

デッドロック検証を行った結果を図 24 に示す。

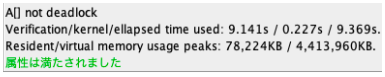

図 24　12 車両で構成される交差点のデッドロック検証結果

6　交差点の通過にかかる最小時間の検証

本節では，車両全てが交差点を 1 回通過するのにかかる最小時間について検証を行う。交差点の使用権の取得方法は前節と同仕様の 5 つ鍵によって管理する。1 回だけなので循環するオートマトンではなく一方通行的なオートマトンを作成する (図 25)。初期状態 start から交差点進入前状態 BeforeEnter への遷移可能条件に交差点の使用権の取得として，L1==0 かつ L2==0 かつ cross==0 を与える。状態 BeforeEnter から交差点通過中状態 crossArea への遷移可能条件と BeforeEnter の状態不変条件で交差点進入前 7 秒以上 10 秒未満の間で使用権を獲得しなければならないかを記述し，状態 crossArea から交差点通過後である終了状態 finish への遷移可能条件と crossArea の状態不変条件で交差点の通過にかかる時間を記述し，crossArea から finish への状態遷移時に使用権の解除を行う。

最小時間の検証を行う。検証のために大域時間変数 gc を宣言する。車両 1 台の start から finish までプロセスにかかる最小時間は 7 秒である。車両は全部で 12 台のため単純に考えると 84 秒で終わるかと考えられるが，同時通行可能なプロセスもあるため，もう少し少ないと考えられる。例えば図 20 のように 4 台同時通行もあり得る。直線が 2 台ずつ，左折が 4 台まとめて，右折が 1 台ずつと考えると，7 台分の時間に短縮されると考え，最小時間は 49 秒と推測し，検証を行う。まず，49 秒で本当に全過程が終わる可能性があるかどうかを次の検証式を用いて検査する。

```
E<> (gc==49
and ns.finish and sn.finish and ew.finish and we.finish
and ne.finish and sw.finish and es.finish and wn.finish
and nw.finish and se.finish and en.finish and ws.finish
```

```
A[] (gc<49 imply
not (ns.finish and sn.finish and ew.finish and we.finis
and ne.finish and sw.finish and es.finish and wn.finish
and nw.finish and se.finish and en.finish and ws.finish
```

1 つ目の最小時間の可能性である上式は満たされたが，最小時間の最小性はを表す下式は，満たされなかった。すなわち，まだ小さい値を取れる。6 台分の時間で検証を引き続き行う。その結果 42 秒が最小であることが求められた。以下の検証式が成り立つ。その時のルートは図 26 の手順だった。

```
E<> (gc==42
and ns.finish and sn.finish and ew.finish and we.finish
```

```
and ne.finish and sw.finish and es.finish and wn.finish
and nw.finish and se.finish and en.finish and ws.finish)

A[] (gc<42 imply
not (ns.finish and sn.finish and ew.finish and we.finish
and ne.finish and sw.finish and es.finish and wn.finish
and nw.finish and se.finish and en.finish and ws.finish))
```

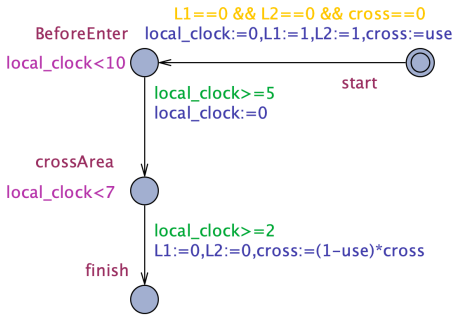

図 25　交差点を 1 回通過する時間オートマトン

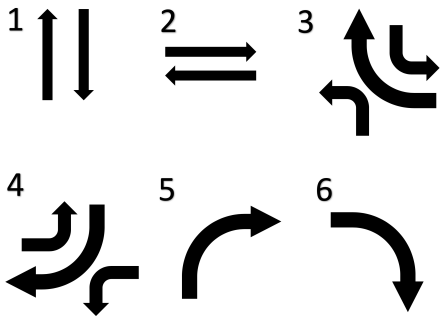

図 26　最小時間のルート

7　おわりに

本研究では，UPPAAL を用いた自動運転車群制御アルゴリズムのモデル化と検証の手法を提案した。単一の交差点においては車両の挙動をモデル化し，デッドロックや通過時間を検証することができた。複数の交差点から構成される都市空間のモデルを作成し検証することが今後の課題である。

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, 近代科学社, 2012.

[2] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella,NuSMV 2: An OpenSource Tool for Symbolic Model Checking,In Proceeding of International Conference on Computer-Aided Verification (CAV 2002), pp. 359-364, 2002.

[3] NuSMV home page, http://nusmv.fbk.eu

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

[5] Spin - Formal Verification, http://spinroot.com

[6] Uppaal in a Nutshell. Kim G. Larsen, Paul Pettersson and Wang Yi. In Springer International Journal of Software Tools for Technology Transfer 1(1+2), 1997.

[7] UPPAAL, http://www.uppaal.org

[8] Timed Automata: Semantics, Algorithms and Tools, Johan Bengtsson and Wang Yi. In Lecture Notes on Concurrency and Petri Nets. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.

[9] 綿引健二, 石川冬樹, 平石邦彦, 時間, 資源の制約をもつビジネスプロセスの形式検証, 電子情報通信学会論文誌 D, 96(8):1878-1891, 2013.