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

電子・情報工学科
中村研究室
佐原優衣

研究背景

- 近年，自動運転技術が急速に発達している。
- 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。

レベル	概要	安全運転に係る監視、対応主体
運転者が一部又は全ての動的運転タスクを実行		
レベル 0 運転自動化なし	• 運転者が全ての動的運転タスクを実行	運転者
レベル 1 運転支援	• システムが縦方向又は横方向のいずれかの車両運動制御のサブタスクを限定領域において実行	運転者
レベル 2 部分運転自動化	• システムが縦方向及び横方向両方の車両運動制御のサブタスクを限定領域において実行	運転者
自動運転システムが（作動時は）全ての動的運転タスクを実行		
レベル 3 条件付運転自動化	• システムが全ての動的運転タスクを限定領域において実行 • 作動継続が困難な場合は、システムの介入要求等に適切に応答	システム (作動継続が困難な場合は運転者)
レベル 4 高度運転自動化	• システムが全ての動的運転タスク及び作動継続が困難な場合への応答を限定領域において実行	システム
レベル 5 完全運転自動化	• システムが全ての動的運転タスク及び作動継続が困難な場合への応答を無制限に（すなわち、限定領域内ではない）実行	システム

官民 ITS 構想・ロードマップ 2018より

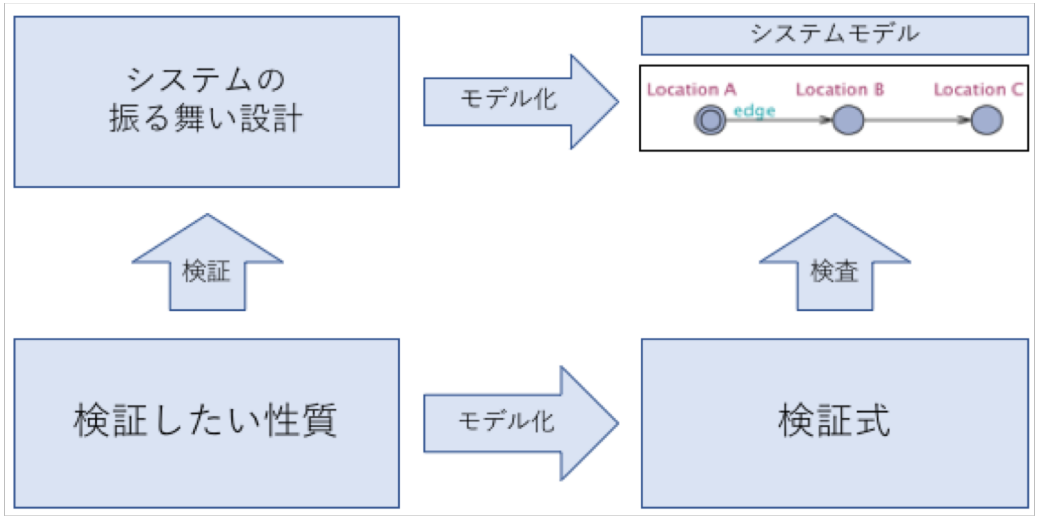
研究背景

- 自動運転車で構成された都市空間において任意の時刻に利用者が自動運転車に乗降し移動するためには大量の車両が必要となる。
- 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。

出典：Masdar社

目的

- 本研究では，自動運転車で構成された都市空間における群制御アルゴリズムをモデル化し，その性質をモデル検査技術によって形式的に検証する手法を提案する。



出典：UPPAALによる性能モデル検証，近代科学社

モデル検査

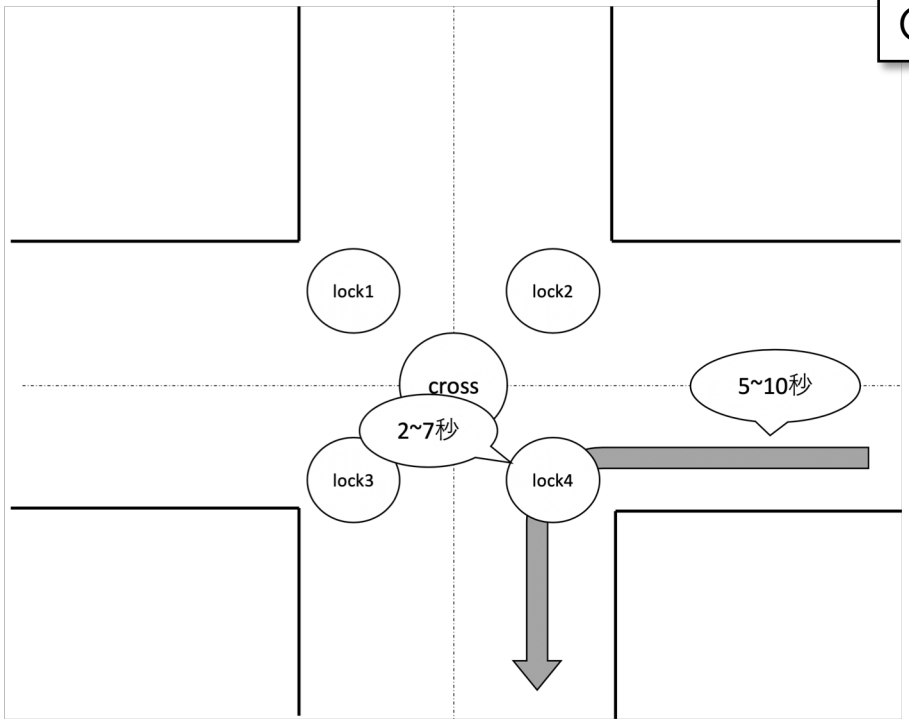
- モデル検査は，システム上で起こり得る状態を網羅的に調べることにより設計の誤りを発見する自動検証手法の一種である。
- 本研究では，時間オートマトンによる時間制約検証が行えるモデル検査ツールUPPAALを採用する。
  - 時間制約問題を扱える
  - 入力がGUIベースのため，直感的に把握できる
  - 検証とGUIによる反例トレース
  - 最短時間で違反状態に到達する反例の出力

本研究のアプローチ

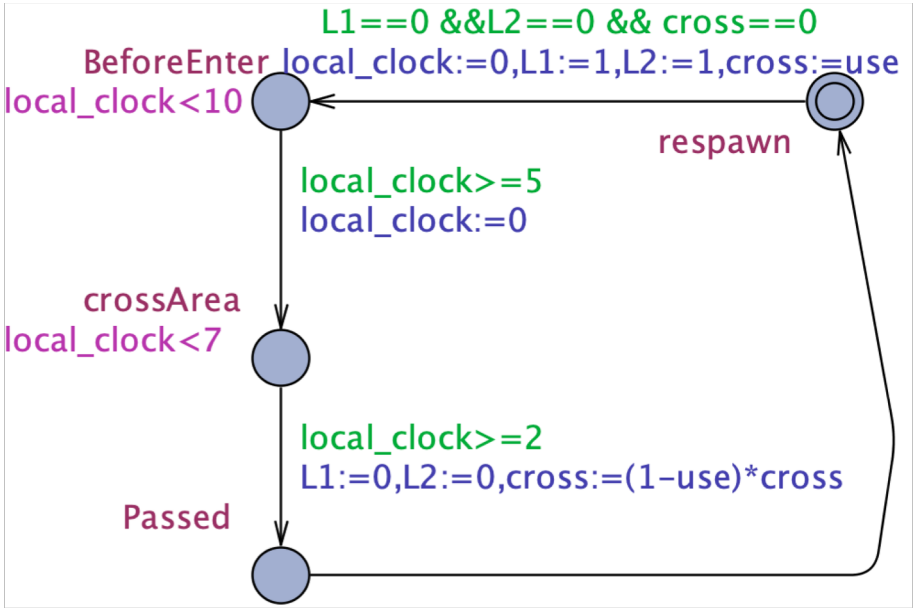
- 交差点通過時の車両モデルを時間オートマトンで記述
- 車両モデルの合成
- シミュレーション
- モデル検査による検証
  - デッドロック検証
  - 最小時間の検証

交差点通過時の車両モデル

- 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
  - 遷移と状態に時間に関する条件を記述する
  - 使用権を大域変数で管理

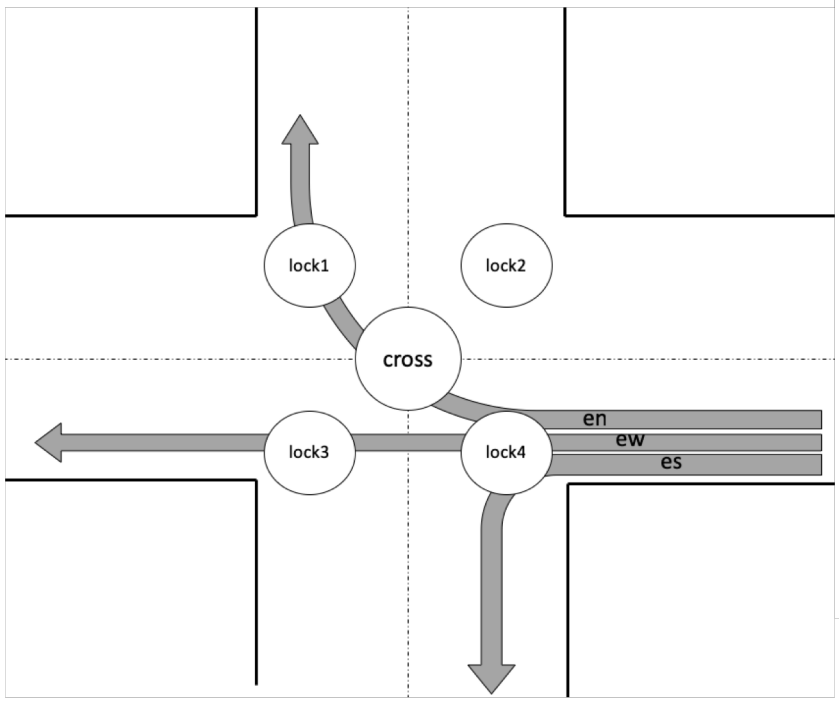


CarCouse(int &L1,int &L2,const int use)



交差点モデル

- 直進，左折，右折の4方向からの進入に対しての車両モデルを作成する（計12台）



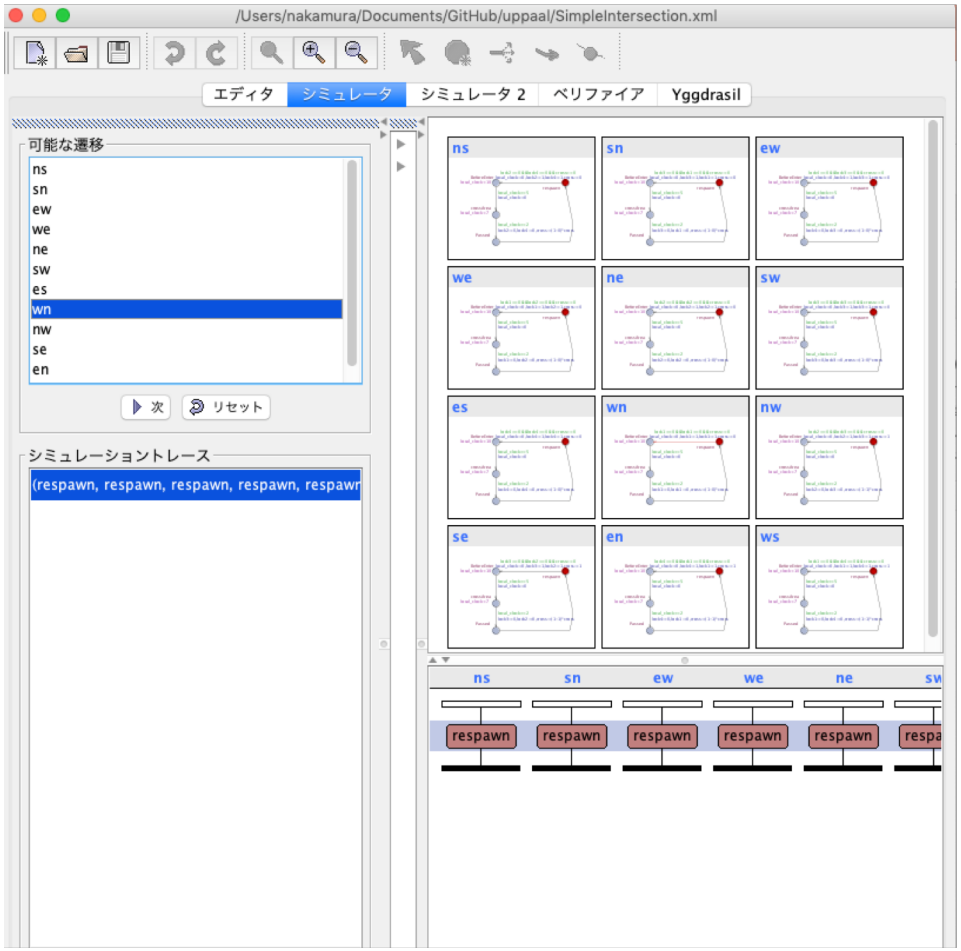

```
// Place template instantiations here.
ns = CarCouse(lock2,lock4,0);
sn = CarCouse(lock3,lock1,0);
ew = CarCouse(lock4,lock3,0);
we = CarCouse(lock1,lock2,0);

ne = CarCouse(lock2,lock2,0);
sw = CarCouse(lock3,lock3,0);
es = CarCouse(lock4,lock4,0);
wn = CarCouse(lock1,lock1,0);

nw = CarCouse(lock2,lock3,1);
se = CarCouse(lock3,lock2,1);
en = CarCouse(lock4,lock1,1);
ws = CarCouse(lock1,lock4,1);
// List one or more processes to be composed
system ns,sn,ew,we,ne,sw,es,wn,nw,se,en,ws;
```

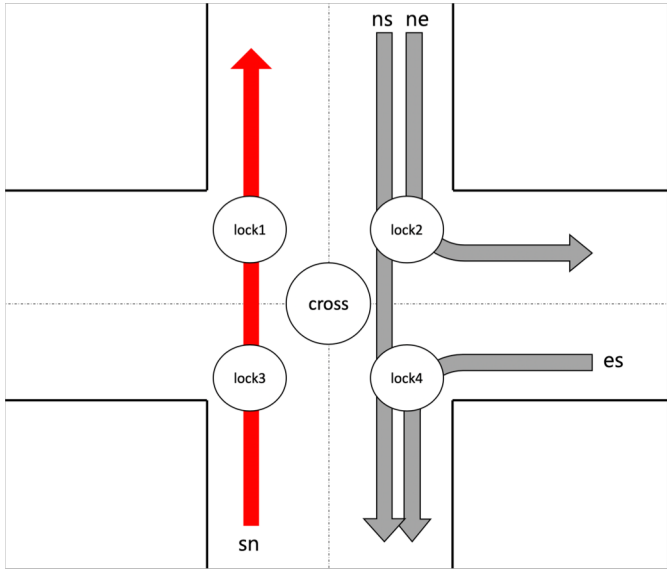
シミュレーション(1/2)

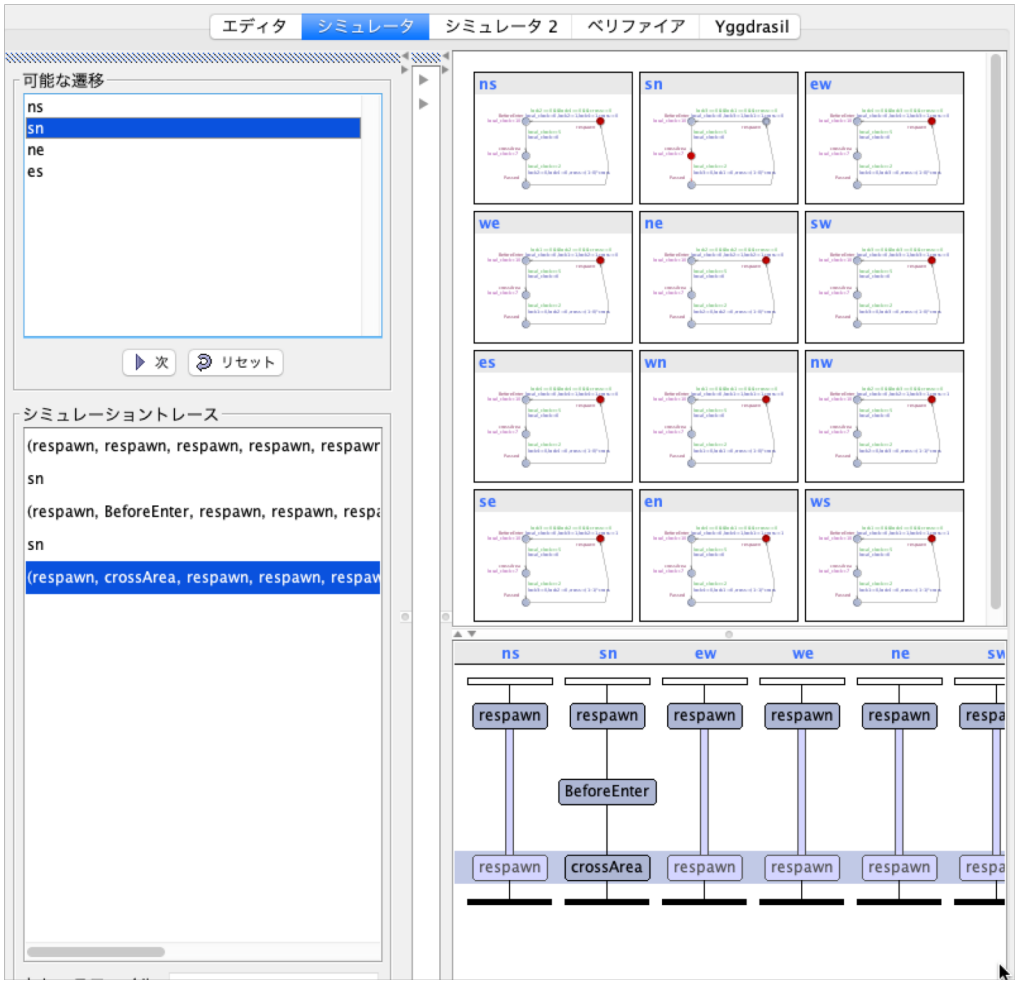
- UPPAALのシミュレーション機能を用いて記述したモデルを確認する
- 初期状態では12車両全ての遷移が可能



シミュレーション(2/2)

- 使用権による交差点に進入可能な車両の制御ができていることが確認できる



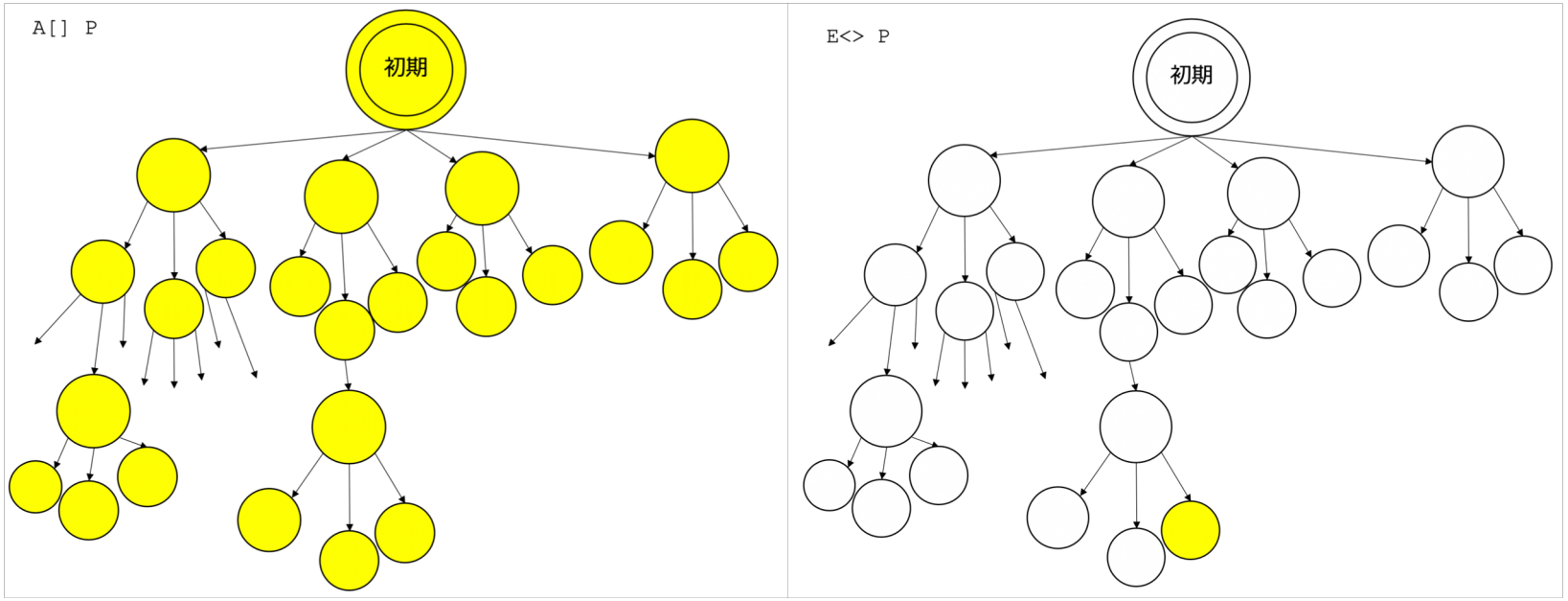


検証

- UPPAALのモデル検査機能を用いて，交差点モデルの性質を検証する

A[] P：全ての実行パスで常に特性Pが成り立つ

E<> P：ある実行パスでいつかは特性Pが成り立つ



デッドロック検証

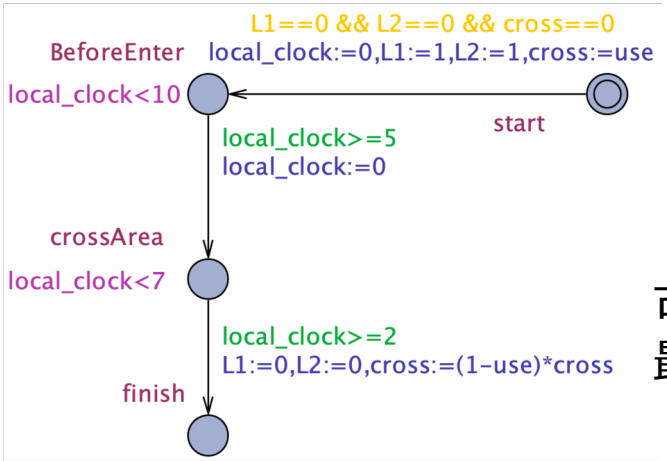
- 記述したモデルでデッドロックが起きないこと検証する
  - デッドロック(deadlock)：どれだけ時間が経過しても，いずれのプロセスも遷移できない場合

```
A[] not deadlock
```

全ての実行パスにおいてデッドロックしないかどうかを検証した

最小時間の検証

- 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



可能性：車両が時間内に通過終了できる
最小性：車両が時間未満では通過終了できない

```
E<> (gc==42
     and ns.finish and sn.finish and ... and ws.finish)

A[] (gc<42 imply not (ns.finish and sn.finish and ...))
```

- 最小時間が42秒であることが検証できた

まとめと今後の課題

- 背景目的に対してどこまでできた？

⇨自分のやったことの位置付け

この鍵の仕組みがデッドロックしないこと

複数の交差点の組み合わせ

左折右折時の時間の計測

交差点のモデル化はできたよ