A Discrete Büchi Automata Distance for Formal Methods Based Control

Garrett Thomas
Supervisor: Lars Lindemann
Examiner: Professor Dimos V. Dimarogonas

Automatic Control Department
Royal Institute of Technology, KTH

June 28$^{\text{th}}$, 2017

Outline

1. Problem and Motivation
   - Formal Methods Based Control
   - The Product Automaton
   - Execution: Path Finding
2. Our Contribution
   - Büchi Distance and Algorithm
3. Performance on Common Formulas
   - Reachability While Avoiding Regions
   - Sequencing
   - Coverage
   - Recurrence (Liveness)
4. More Complex Formulas
   - Study of Various Formulas
5. Conclusions

Outline

Linear Temporal Logic

- We will be using **Linear Temporal Logic (LTL)**, defined recursively as $\varphi ::= \top|\alpha|\neg\varphi_1|\varphi_1 \vee \varphi_2|\mathbf{X}\varphi_1|\varphi_1\boldsymbol{\mathcal{U}}\varphi_2$

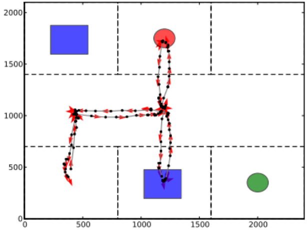
Linear Temporal Logic

- We will be using **Linear Temporal Logic (LTL)**, defined recursively as $\varphi ::= \top|\alpha|\neg\varphi_1|\varphi_1 \vee \varphi_2|\mathbf{X}\varphi_1|\varphi_1\boldsymbol{\mathcal{U}}\varphi_2$

Ex. from [Guo15a]

$\varphi = \diamond(\text{rball} \wedge \diamond\text{basket}) \wedge \diamond\square\text{r1}$

"Eventually pick up the red ball and put it in one of the baskets. Then go home to r1"



LTL Motion and Task Planning

- LTL Motion and task planning is done in three Steps [BBE+07]

LTL Motion and Task Planning

- LTL Motion and task planning is done in three Steps [BBE$^+$07]
- Specification: Create a graph (the *product automaton*)
  - Workspace, robot motion, and LTL formula

LTL Motion and Task Planning

- LTL Motion and task planning is done in three Steps [BBE+07]
- Specification: Create a graph (the *product automaton*)
  - Workspace, robot motion, and LTL formula
- Execution: Find a discrete path in this graph using an optimality criterion.

LTL Motion and Task Planning

- LTL Motion and task planning is done in three Steps [BBE+07]
- Specification: Create a graph (the *product automaton*)
  - Workspace, robot motion, and LTL formula
- Execution: Find a discrete path in this graph using an optimality criterion.
- Implementation: Calculate the continuous controllers such that the continuous path will satisfy the discrete path.

LTL Motion and Task Planning

- LTL Motion and task planning is done in three Steps [BBE$^+$07]
- Specification: Create a graph (the *product automaton*)
  - Workspace, robot motion, and LTL formula
- **Execution: Find a discrete path in this graph using an optimality criterion.**
- Implementation: Calculate the continuous controllers such that the continuous path will satisfy the discrete path.

Finite-State Transition System

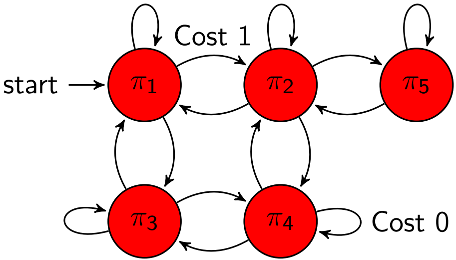
- The product automaton is the combination of a *finite-state transition system* and a *Büchi automaton*

Finite-State Transition System

- The product automaton is the combination of a *finite-state transition system* and a *Büchi automaton*

Finite-State Transition System (FTS)

An FTS is a tuple $\mathcal{T} = (\Pi, \rightarrow, \Pi_0, AP, L_D)$ where Π is the set of states, $\rightarrow \subseteq \Pi \times \Pi$ is the transitions, $\Pi_0 \subseteq \Pi$ is the initial state(s), AP is the set of atomic propositions, and $L : \Pi \rightarrow 2^{AP}$ is the labelling function (goes from a state to the set of atomic propositions that are true in that state).



Büchi Automaton

Büchi Automaton

A Büchi automaton is a tuple $\mathcal{A}_\varphi = (\mathcal{Q}, 2^{AP}, \delta, \mathcal{Q}_0, \mathcal{F})$ where $\mathcal{Q}$ is a finite set of states, $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is the set of initial states, 2^{AP} is the alphabet, $\delta : \mathcal{Q} \times 2^{AP} \to 2^{\mathcal{Q}}$ is a transition relation, and $\mathcal{F} \subseteq \mathcal{Q}$ is the set of accepting states.

- A path on a Büchi automaton is accepting if it passes through an accepting state infinitely many times.
- For any LTL formula φ over AP, there exists a Büchi automaton over 2^{AP} corresponding to φ [BKL08]

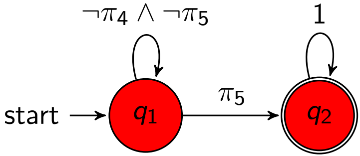
Büchi Automaton

Büchi Automaton

A Büchi automaton is a tuple $\mathcal{A}_\varphi = (\mathcal{Q}, 2^{AP}, \delta, \mathcal{Q}_0, \mathcal{F})$ where $\mathcal{Q}$ is a finite set of states, $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is the set of initial states, 2^{AP} is the alphabet, $\delta : \mathcal{Q} \times 2^{AP} \to 2^{\mathcal{Q}}$ is a transition relation, and $\mathcal{F} \subseteq \mathcal{Q}$ is the set of accepting states.

- A path on a Büchi automaton is accepting if it passes through an accepting state infinitely many times.
- For any LTL formula φ over AP, there exists a Büchi automaton over 2^{AP} corresponding to φ [BKL08]

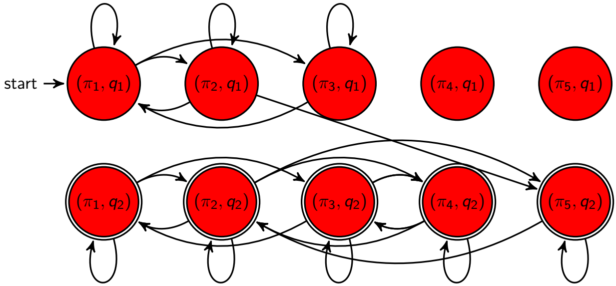
Reachability while avoiding regions $\varphi = \neg\pi_4 \boldsymbol{\mathcal{U}} \pi_5$



Product Automaton

$\mathcal{A}_p = \mathcal{T}_w \otimes \mathcal{A}_\varphi = (Q', \delta', Q'_0, \mathcal{F}')$, where
$Q' = \Pi \times Q = \{\langle \pi, q \rangle \in Q' | \forall \pi \in \Pi, \;\; \forall q \in Q\}$; $\delta' : Q' \rightarrow 2^{Q'}$.
$\langle \pi_j, q_n \rangle \in \delta'(\langle \pi_i, q_m \rangle)$ iff $(\pi_i, \pi_j) \in \rightarrow_c$ and $q_n \in \delta(q_m, L_d(\pi_j))$;
$Q'_0 = \{\langle \pi, q \rangle | \pi \in \Pi_0, \;\; q_0 \in Q_0\}$, $\mathcal{F}' = \{\langle \pi, q \rangle | \pi \in \Pi, q \in \mathcal{F}\}$

- Also a Büchi automaton



Accepted Algorithm

- We prefer a prefix, suffix structure
  $R = \langle R_{pre}, R_{suf} \rangle = q'_0 q'_1 \dots q'_f [q'_{f+1} q'_{f+2} \dots q'_n q'_f]^\omega$

Here is the algorithm currently used in the literature [Guo15a],[FGKGP09],[KB08],[STBR10]

Procedure 1 OptRun() [Guo15a]

Input: Input $\mathcal{A}_p$

Output: R_{opt}

1. For initial state $q'_0 \in \mathcal{Q}'_0$, find the optimal path to each $q'_f \in \mathcal{F}$.
2. For each accepting state $q'_f \in \mathcal{F}'$, calculate the optimal path back to q'_f.
3. Find the pair of $(q'_{0,opt}, q'_{f,opt})$ that minimizes the total cost
4. Optimal accepting run R_{opt}, prefix: shortest path from q'_{0*} to q_{f*}; suffix: the shortest cycle from q'_{f*} and back to itself.

State-Space Explosion Problem

- State-space explosion problem is the combinatorial explosion of the number of states in the product automaton.
- Number of states in Büchi automaton can be exponential in the size of the LTL formula [GL02] and $|\mathcal{Q}'| = |\Pi| \cdot |\mathcal{Q}|$
- State-space explosion problem is the bottle neck of formal methods based control synthesis.

Outline

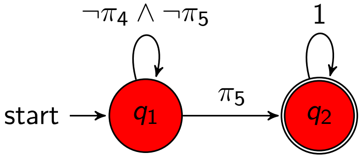
Büchi Distance Measure

- We introduce a discrete Büchi distance measure, which is the least number of transitions possible to get to an accepting state

Büchi Distance Measure

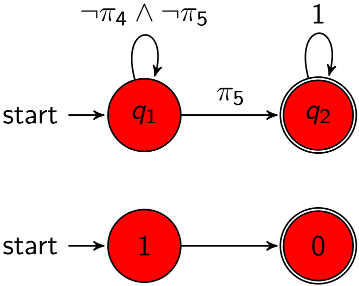
- We introduce a discrete Büchi distance measure, which is the least number of transitions possible to get to an accepting state

Ex. Reachability While Avoiding Regions $\neg\pi_4 \mathcal{U} \pi_5$



- We introduce a discrete Büchi distance measure, which is the least number of transitions possible to get to an accepting state

Ex. Reachability While Avoiding Regions $\neg\pi_4 \mathcal{U} \pi_5$



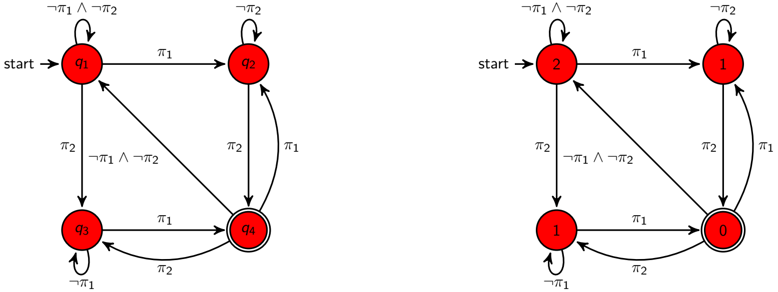
- Denoted $d_p : \mathcal{Q} \to \mathbb{Z}$, e.g. $d_p(q_2) = 1$

Note: Transitions with && are removed from LTL2BA because regions do not overlap

Büchi Distance Measure Cont.

Büchi Automaton with Distance

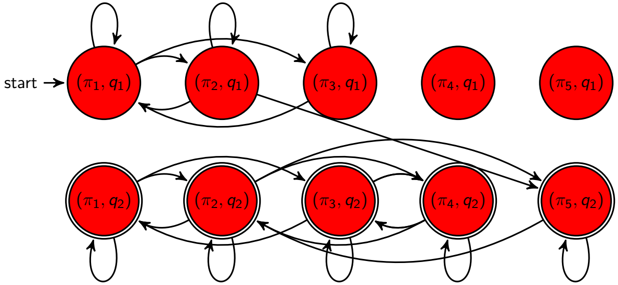
A Büchi automaton with distance $\mathcal{A}_{\varphi,d} = (\mathcal{Q}, 2^{AP}, \delta, \mathcal{Q}_0, \mathcal{F}, d)$ where $\mathcal{Q}, 2^{AP}, \delta, \mathcal{Q}_0, \mathcal{F}$ are defined as before and $d : \mathcal{Q} \to \mathbb{Z}$ is defined as $d(q_n) = \min_i\{|q_i \in \mathcal{F}$ and $q_k \in \delta(q_{k-1}, S_{k-1})$ for some $S_k \in 2^{AP}$ and $k = 0, 1, \ldots, i-1\}$



Büchi Distance in Product Automaton

Product Automaton with Distance

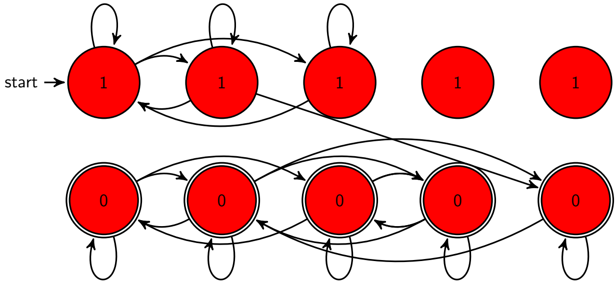
Product automaton with distance, $\mathcal{A}_{p,d} = \mathcal{T} \otimes \mathcal{A}_\varphi = (Q', \delta', Q'_0, \mathcal{F}', d_p)$, defined similarly, with $d_p(q') = d(q'|\mathcal{Q})$



Büchi Distance in Product Automaton

Product Automaton with Distance

Product automaton with distance, $\mathcal{A}_{p,d} = \mathcal{T} \otimes \mathcal{A}_\varphi = (Q', \delta', Q'_0, \mathcal{F}', d_p)$, defined similarly, with $d_p(q') = d(q'|\mathcal{Q})$



Greedy Algorithm

- Motivation for our algorithm: $\mathcal{F}' = \{\langle \pi, q \rangle | \pi \in \Pi, q \in \mathcal{F}\}$
- We greedily find the optimal path which reduces the Büchi distance at each step

Procedure 2 GreedyRun()

Input: Input $\mathcal{A}_{p,d}$
Output: R_g

1: LEVEL $= d_p(q'_0 \in \mathcal{Q}'_0)$
2: **while** LEVEL > 0 **do**
3: find optimal path down to q'_n s.t. $d_p(q'_n) == LEVEL - 1$
4: Level = Level - 1
5: Find optimal path from q'_n back to itself
6: Accepting run R_g, prefix: the optimal paths calculated in the while loop concatenated together; suffix: optimal path from q'_n back to itself.

Why?

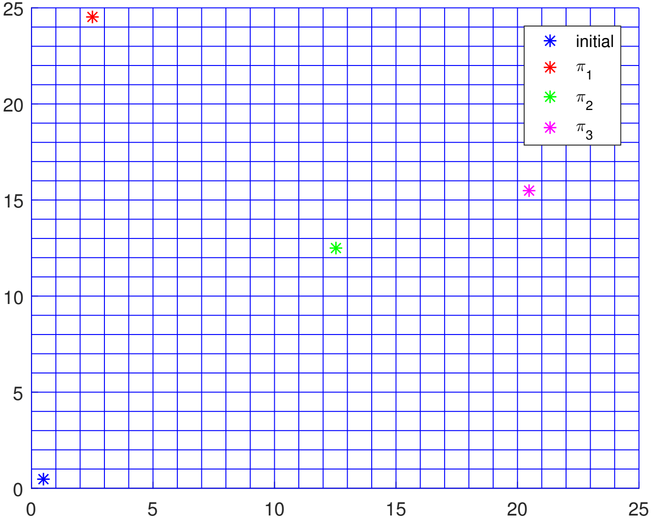
- We approximate the globally optimal path with a series of locally optimal paths.
- We sacrifice a degree of optimality for easier computation!

Code

- The code for the accepted algorithm is from the P_MAS_TG GitHub Repository [Guo15b].
- The code for the greedy algorithm is a modified version of code from P_MAS_TG.
- All computations were done on a 2.5 GHz MacBook Pro and used Python 2.7.5.

Outline

Workspace



Reachability While Avoiding Regions

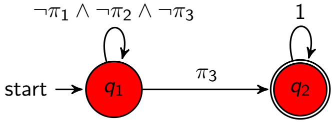


Figure: Büchi automaton corresponding to $\neg(\pi_1 \vee \pi_2)\mathcal{U}\pi_3$

$$d_p(q_1) = 1 \text{ and } d_p(q_2) = 0$$

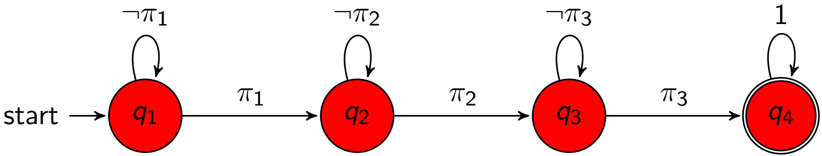
```
Accepted Algorithm
=====================
plan done within 0.02s: precost 37.00, sufcost 0.00
...
full construction and synthesis done within 0.11s

Greedy Algorithm
=====================
plan done within 0.01s: precost 37.00, sufcost 0.00
...
full construction and synthesis done within 0.10s
```

Animation

Sequencing

- We look at the sequencing formula $\diamond(\pi_1 \wedge \diamond(\pi_2 \wedge \diamond\pi_3))$



- $d_p(q_1) = 3,\ d_p(q_2) = 2,\ d_p(q_3) = 1,\ d_p(q_4) = 0$
- Only one path down $\rightarrow$ Both algorithms calculate the same path!

Simulation

```
Accepted Algorithm
==================
plan done within 0.04s: precost 62.00, sufcost 0.00
...
full construction and synthesis done within 0.19s
```

Our algorithm computed the same path, with an output of

```
Greedy Algorithm
==================
plan done within 0.02s: precost 62.00, sufcost 0.00
...
full construction and synthesis done within 0.17s
```

Nodes Searched

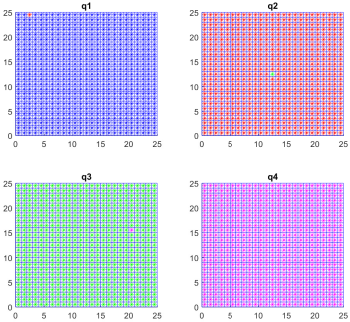


Figure: Nodes with accepted algorithm

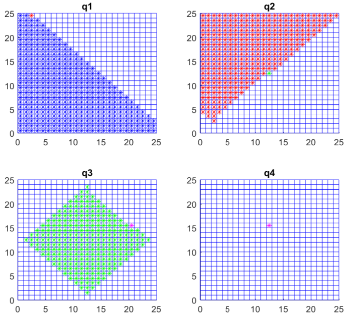


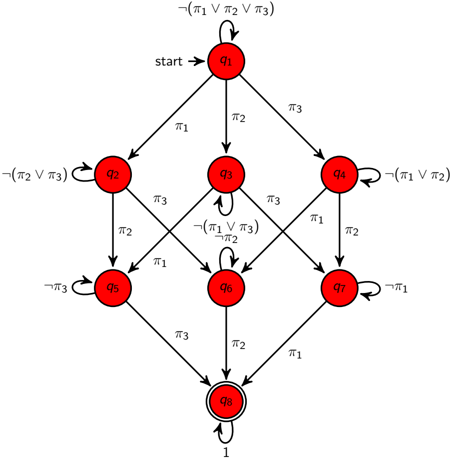
Figure: Nodes with greedy algorithm

Coverage

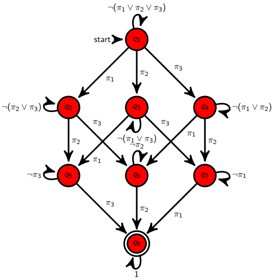
- $\varphi = \diamond\pi_1 \wedge \diamond\pi_2 \wedge \cdots \wedge \diamond\pi_n$.
- A coverage formula represents the statement visit $\pi_1, \pi_2, \ldots, \pi_n$ in any order. Ex. $\varphi = \diamond\pi_1 \wedge \diamond\pi_2 \wedge \diamond\pi_3$

Coverage

- $\varphi = \diamond\pi_1 \wedge \diamond\pi_2 \wedge \cdots \wedge \diamond\pi_n$.
- A coverage formula represents the statement visit $\pi_1, \pi_2, \ldots, \pi_n$ in any order. Ex. $\varphi = \diamond\pi_1 \wedge \diamond\pi_2 \wedge \diamond\pi_3$



Coverage



- Now we have "choices" that have to be made!
- Choices → we lose optimality. The path our algorithm finds will likely cost more than the path calculated by the accepted algorithm. However our path is computed faster.

Coverage

```
Accepted Algorithm
==================
plan done within 0.08s: precost 59.00, sufcost 0.00
...
full construction and synthesis done within 0.43s
```

and our algorithm is

```
Greedy Algorithm
================
plan done within 0.02s: precost 62.00, sufcost 0.00
...
full construction and synthesis done within 0.38s
```

Cost Bound

- The travelling salesperson problem: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

Cost Bound

- The travelling salesperson problem: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

Greedy Algorithm Cost Bound

$$\text{GREEDY} + 2\max_{i,j} c_{i,j} \leq \left(\frac{1}{2}\lceil \log(n) \rceil + \frac{1}{2}\right)(\text{ACCEPT} + 2\max_{i,j} c_{i,j})$$

Recurrence (Liveness)

- Recurrence: "Visit π_1, π_2, ..., π_n infinitely many times."

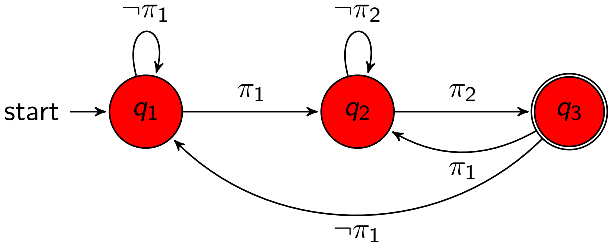


Figure: Büchi Automaton for $\Box(\Diamond\pi_1 \land \Diamond\pi_2)$ 1

- Automaton from LTL2BA. Note: Not tight.

Recurrence (Liveness)

- Recurrence: "Visit π_1, π_2, ..., π_n infinitely many times."

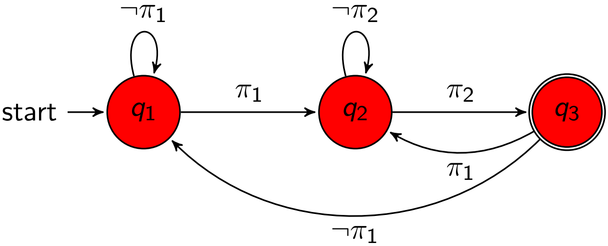


Figure: Büchi Automaton for $\Box(\Diamond\pi_1 \wedge \Diamond\pi_2)$ 1

- Automaton from LTL2BA. Note: Not tight.
- For the first time we have a formula that has a **non-trivial suffix**.

Case Study

- Remember that the accepted algorithm computes the suffix from **every** accepting state. That implies a lot of work for this formula.

Case Study

- Remember that the accepted algorithm computes the suffix from **every** accepting state. That implies a lot of work for this formula.

```
Accepted Algorithm
==========================
plan done within 16.17s: precost 62.00, sufcost 60.00
----------------------------------------

...
full construction and synthesis done within 16.35s
```

while our algorithm did it in

```
Greedy Algorithm
==========================
plan done within 0.04s: precost 62.00, sufcost 60.00
----------------------------------------

...
full construction and synthesis done within 0.21s
```

Outline

- More complex formulas prove difficult to analyze because the Büchi automaton becomes very big and cannot be visualized.

Formula	Accepted Cost	Accepted Time	Greedy Cost	Greedy Time
'(!r223 U r445) || (!r268 U r435)'	27	0.04	27	0.01
'!r62 U(!r266 U r422)'	38	0.05	38	0.02
'[]<> r0 − > []<> r317'	1	5.06	1	0.00
'[]<> r0 < − > []<> r317'	1	10.70	1	0.00
'!(<><> r498 < − > r541)'	42	0.03	42	0.02
'!([]<> r3 − > []<>r591)'	3	5.06	3	0.00
'!([]<> r3 < − > []<>r591)'	3	10.31	39	0.01
'!r532 R (!r432 || r321)'	0	4.97	0	0.01
'<> r114 && [](r114 − ><> r12) && ((X r114 U X r12) || !X(r114 U r12))'	24	0.08	24	0.01
'<> pickrball && [](pickrball − ><> droprball) && ((X pickrball U X droprball) || !X(pickrball U droprball))'	47	28.87	47	0.03
' <> r124 && <> !r124'	28	0.05	28	0.01

Table: Comparison of Accepted Algorithm with Greedy Algorithm

Outline

Benefits

- Works very well on reachability while avoiding regions, sequencing, and recurrence (when the automaton is not tight). Guaranteed to get the same path faster!
- Saves a lot of time when the formula does not have a trivial suffix.

Considerations

- Hard to analyze the performance on more complex formulas.
- When there is a trivial suffix, the majority of the time is spent on constructing the graph and the search is usually quick. Future work could be to use this algorithm for on-the-fly construction.

References I

Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins, and George J Pappas, *Symbolic planning and control of robot motion [grand challenges of robotics]*, IEEE Robotics & Automation Magazine **14** (2007), no. 1, 61–70.

Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen, *Principles of model checking*, MIT press, 2008.

Georgios E Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J Pappas, *Temporal logic motion planning for dynamic robots*, Automatica **45** (2009), no. 2, 343–352.

Dimitra Giannakopoulou and Flavio Lerda, *From states to transitions: Improving translation of ltl formulae to büchi automata*, Formal Techniques for Networked and Distributed SytemsFORTE 2002 (2002), 308–326.

Meng Guo, *Hybrid control of multi-robot systems under complex temporal tasks*, Ph.D. thesis, KTH Royal Institute of Technology, 2015.

Meng Guo, *P-mas-tg*, `https://github.com/MengGuo/P_MAS_TG`, 2015.

Marius Kloetzer and Calin Belta, *A fully automated framework for control of linear systems from temporal logic specifications*, IEEE Transactions on Automatic Control **53** (2008), no. 1, 287–297.

Stephen L Smith, Jana Tumova, Calin Belta, and Daniela Rus, *Optimal path planning under temporal logic constraints*, Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, IEEE, 2010, pp. 3288–3293.

THANK YOU!

Thanks for listening, any questions? :)