STyLuS*: A Temporal Logic Optimal Control Synthesis Tool for Large-Scale Multi-Robot Systems

Yiannis Kantaros, Michael M. Zavlanos

I. Setting Up STyLuS*

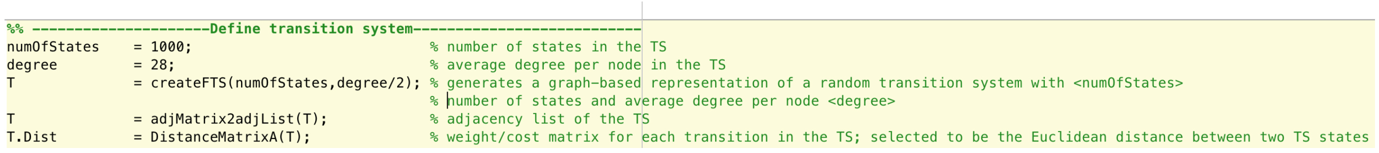
A. Dependencies

STyLuS has been tested on Windows 7 and on macOS HighSierra on MATLab R2016b. To use STyLuS, the ltl2ba toolbox needs to be installed that is available online on http://www.lsv.fr/~gastin/ltl2ba/. Once the executable file is generated, it should be placed in the folder `functions`. Executable files generated on Windows 7 (ltl2baWin7) and macOS HighSierra (ltl2ba) are already included there. The user is encouraged to generate new executable files to avoid any inconsistencies. The ltl2ba toolbox is used in the function `create_buchi` to translate LTL formulas into Non-deterministic Bucchi automata (NBA). Note that the current MATLAB folder should be `.../functions/` (i.e., where the executable file is located), otherwise `create_buchi` will not be able to generate a NBA.

B. Inputs

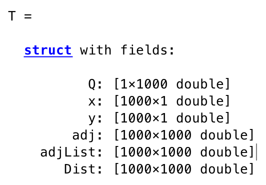
The user-specified inputs to STyLuS are determined in the function `DefineInputs.m`:

- a weighted Transition System (TS) modeling robot mobility; see also Figs. 1-2. STyLuS has currently been implemented assuming that all robots have the same model, i.e., the same TS. A TS is represented by a structure `T`. The user needs to define the (i) number of TS states (`numStates`) and the TS state-space defined as $\texttt{T.Q} = [1 : \texttt{numStates})]$, (ii) the physical locations of the TS states, i.e., the x (vector `T.x` with dimensions $\texttt{numStates} \times 1$) and y (vector `T.y` with dimensions $\texttt{numStates} \times 1$) coordinate of each state, and (iii) the adjacency matrix of the TS (`T.adj` which is a $\texttt{numStates} \times \texttt{numStates}$ matrix) where the entry $\texttt{T.adj}(i, j)$ is 0 if a direct transition from state i to state j is infeasible/possible and 1 otherwise. By default, STyLuS generates these three inputs randomly. If a user-specified TS is needed, the command `T=createFTS(numOfStates,degree/2)` should be commented out. The field `T.Dist` is a $\texttt{numStates}x\texttt{numStates}$ capturing the cost for each TS transition that will be generated automatically. `T.Dist` is defined so that it captures the Euclidean distance from the TS state i to the TS state j if it holds $\texttt{T.adj}(i, j) = 1$; for all other transitions for which $\texttt{T.adj}(i, j) = 0$, it holds that $\texttt{T.Dist}(i, j) = \texttt{Inf}$. The user can comment out the command `T.Dist = DistanceMatrixA(T)` and assign manually new weights to each transition.



```
%% --------------------Define transition system--------------------------
numOfStates     = 1000;                         % number of states in the TS
degree          = 28;                           % average degree per node in the TS
T               = createFTS(numOfStates,degree/2); % generates a graph-based representation of a random transition system with <numOfStates>
                                                % number of states and average degree per node <degree>
T               = adjMatrix2adjList(T);         % adjacency list of the TS
T.Dist          = DistanceMatrixA(T);           % weight/cost matrix for each transition in the TS; selected to be the Euclidean distance between two TS states
```

Fig. 1. A weighted Transition System as an input to STyLuS*.

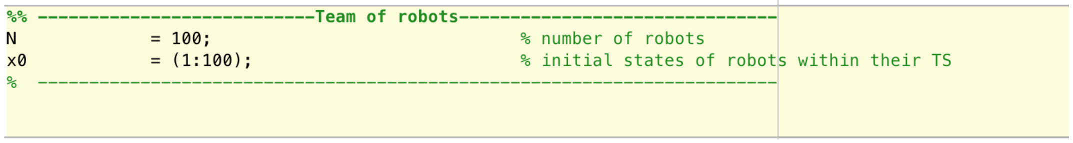


```
T =

  struct with fields:

          Q: [1×1000 double]
          x: [1000×1 double]
          y: [1000×1 double]
        adj: [1000×1000 double]
    adjList: [1000×1000 double]
       Dist: [1000×1000 double]
```

Fig. 2. Structure of a weighted Transition System as an input to STyLuS*.

- number `N` of robots and their initial state `x0`; see also Fig. 3. `x0` is an $\texttt{N} \times 1$ vector where the i-th entry captures the initial state of robot i and has to be an integer within the interval $[1, \texttt{numStates}]$.
- an LTL mission captured in the string phi; see also Fig. 4. To define the LTL formula phi, the following notation is used: G, F, U, X stand for the 'always', 'eventually', 'until', and 'next' operator, and $\&, |, !$ stand for the logical 'and', 'or', and 'not' operators. The number of atomic predicates p that appear in phi needs to be defined as well (see `Np`). For instance,

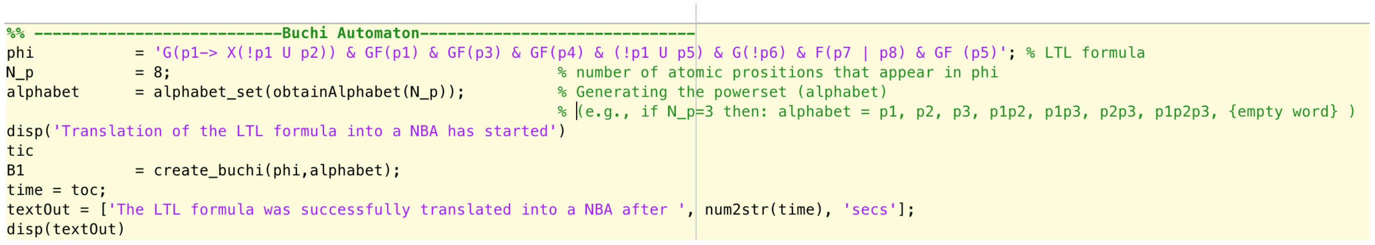
$$phi = F(p1)\ \&\ G(!p2)$$



```
%% ----------------------------Team of robots-------------------------------
N              = 100;                           % number of robots
x0             = (1:100);                       % initial states of robots within their TS
%  -------------------------------------------------------------------------
```

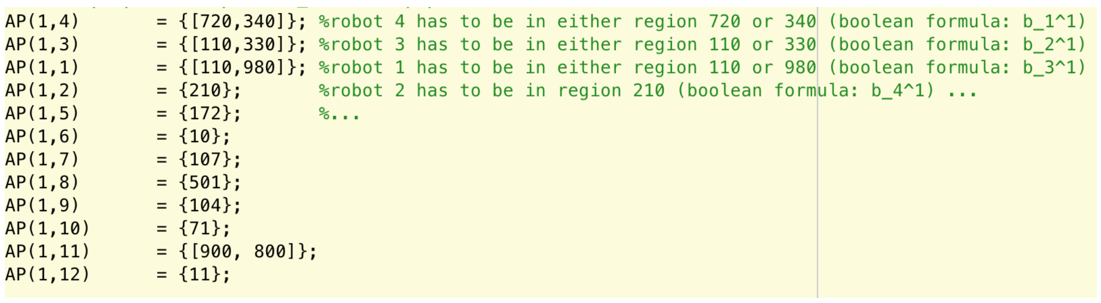
Fig. 3. Initialization of a team of N robots.

means that eventually $p1$ needs to satsified and $p2$ should never be satisfied. The definition of where the predicates (e.g., $p1$ and $p2$) in phi are satisfied is captured by the cell array AP; see also Fig. 5. Let pk denote the k-th predicate which is captured by $AP(k,:)$. Specifically, $AP(k,r)$ is an empty cell if robot r does not play any role in satisfying pk and a non-empty cell array otherwise. If it is a non-empty cell, then $AP(k,r)$ is defined as $AP(k,r) = \{[\text{state } 1, \text{state } 2, \dots, \text{state m}]\}$ collecting all states that if at least one of them is visited by robot r then $AP(k,r)$ is true. Note that this definition allows more than one robot to contribute to satisfaction of pk. For instance, if pk is defined to be true if robots either robot $r1$ or robot $r2$, or robot $r3$ are in state m, then pk is defined as follows $AP(k,r1) = \{m\}$, $AP(k,r2) = \{m\}$, $AP(k,r3) = \{m\}$. Also, to capture the logical 'or' in the definition of pk, we set $AP(k,N+3) = \{1\}$. Otherwise, if $AP(k,N+3) = \{0\}$, it means that all three robots have to be in state m so that pk is true.



```
%% ---------------------------Buchi Automaton------------------------------
phi           = 'G(p1-> X(!p1 U p2)) & GF(p1) & GF(p3) & GF(p4) & (!p1 U p5) & G(!p6) & F(p7 | p8) & GF (p5)'; % LTL formula
N_p           = 8;                                          % number of atomic prositions that appear in phi
alphabet      = alphabet_set(obtainAlphabet(N_p));          % Generating the powerset (alphabet)
                                                            % (e.g., if N_p=3 then: alphabet = p1, p2, p3, p1p2, p1p3, p2p3, p1p2p3, {empty word} )
disp('Translation of the LTL formula into a NBA has started')
tic
B1            = create_buchi(phi,alphabet);
time = toc;
textOut = ['The LTL formula was successfully translated into a NBA after ', num2str(time), 'secs'];
disp(textOut)
```

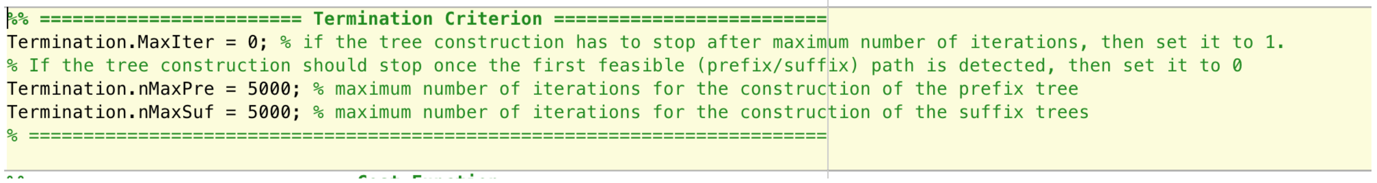
Fig. 4. An LTL formula as an input to STyLuS*.



```
AP(1,4)       = {[720,340]}; %robot 4 has to be in either region 720 or 340 (boolean formula: b_1^1)
AP(1,3)       = {[110,330]}; %robot 3 has to be in either region 110 or 330 (boolean formula: b_2^1)
AP(1,1)       = {[110,980]}; %robot 1 has to be in either region 110 or 980 (boolean formula: b_3^1)
AP(1,2)       = {210};       %robot 2 has to be in region 210 (boolean formula: b_4^1) ...
AP(1,5)       = {172};       %...
AP(1,6)       = {10};
AP(1,7)       = {107};
AP(1,8)       = {501};
AP(1,9)       = {104};
AP(1,10)      = {71};
AP(1,11)      = {[900, 800]};
AP(1,12)      = {11};
```

Fig. 5. Determining when $p1$ is satisfied within STyLuS*.

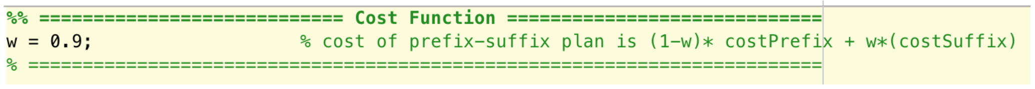
- Termination criterion (`Termination`);see also Fig. 6. The user needs to define if STyLuS should terminate once the first feasible solution is found (`Termination.MaxIter=0`) or after a maximum number of iterations (`Termination.MaxIter=1`). In the later case, the user needs to determine the maximum number of iterations for the construction of the prefix (`Termination.nMaxPre`) and suffix (`Termination.nMaxSuf`) plan.



```
%% ======================== Termination Criterion ========================
Termination.MaxIter = 0; % if the tree construction has to stop after maximum number of iterations, then set it to 1.
% If the tree construction should stop once the first feasible (prefix/suffix) path is detected, then set it to 0
Termination.nMaxPre = 5000; % maximum number of iterations for the construction of the prefix tree
Termination.nMaxSuf = 5000; % maximum number of iterations for the construction of the suffix trees
% =======================================================================
```

Fig. 6. Determining the termination criterion of STyLuS*.

- a parameter $\mathtt{w} \in [0,1]$ required to determine the cost of a prefix suffix plan; see also Fig. 7. In particular the cost of a prefix-suffix plan is $(1-w)J(\mathtt{prefix})+wJ(\mathtt{suffix})$, where the cost of the prefix and suffix (i.e., $J(\mathtt{prefix}), J(\mathtt{suffix})$ is determined by `T.Dist`.



```
%% ============================ Cost Function =============================
w = 0.9;                       % cost of prefix-suffix plan is (1-w)* costPrefix + w*(costSuffix)
% =========================================================================
```

Fig. 7. Determining the cost-related parameter w within STyLuS*.

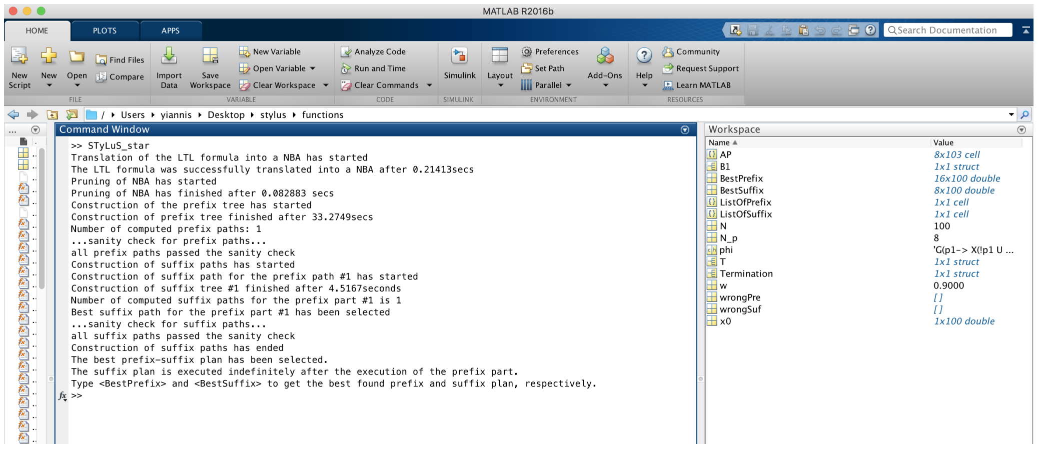
C. Outputs

The output of STyLuS is generated by the function `MainSTyLuS_star` and includes the following:

- the best found prefix-suffix plan (`BestPrefix`, `BestSuffix`). The prefix part is defined as an array with dimensions $H \times N$ where H captures the number or waypoints that the multi robot system has to take. To get the full prefix path, type in the command window `BestPrefix`. To get k-th waypoint of robot j, type in the command window `BestPrefix(k,j)`. The same holds for the corresponding suffix part `BestSuffix`.
- a cell array containing all the detected prefix parts (`ListOfPrefix`) including the best one. To get the m-th prefix part, type in the command window: `ListOfPrefixm` . To get k-th waypoint of robot j type in the m-th prefix part, type in the command window `ListOfPrefixm(k,j)` . The same holds for the cell array containing all the detected suffix parts (`ListOfSuffix`)
- a vector (`wrongPre`) containing the indices of the detected prefix parts that did not pass sanity checks (e.g., a prefix path has to respect the transition rule of the TS). If any prefix part does not pass the sanity check a warning message will appear on the command window and the index to these prefix parts will be included in this vector. A corresponding vector for the suffix parts is also returned (`wrongSuf`).

D. Execution

To execute `STyLuS*`, type `STyLuS_star` in the MATLab command window. The output of STyLuS* for a case study with $N = 100$ robots where each robot is modeled as TS with 1000 states and collaborative LTL formula corresponding to an NBA with 21 states is shown in Fig. 8.



```
>> STyLuS_star
Translation of the LTL formula into a NBA has started
The LTL formula was successfully translated into a NBA after 0.21413secs
Pruning of NBA has started
Pruning of NBA has finished after 0.082883 secs
Construction of the prefix tree has started
Construction of prefix tree finished after 33.2749secs
Number of computed prefix paths: 1
...sanity check for prefix paths...
all prefix paths passed the sanity check
Construction of suffix paths has started
Construction of suffix path for the prefix path #1 has started
Construction of suffix tree #1 finished after 4.5167seconds
Number of computed suffix paths for the prefix part #1 is 1
Best suffix path for the prefix part #1 has been selected
...sanity check for suffix paths...
all suffix paths passed the sanity check
Construction of suffix paths has ended
The best prefix-suffix plan has been selected.
The suffix plan is executed indefinitely after the execution of the prefix part.
Type <BestPrefix> and <BestSuffix> to get the best found prefix and suffix plan, respectively.
>>
```

Name	Value
AP	8x103 cell
B1	1x1 struct
BestPrefix	16x100 double
BestSuffix	8x100 double
ListOfPrefix	1x1 cell
ListOfSuffix	1x1 cell
N	100
N_p	8
phi	'G(p1-> X(!p1 U ...
T	1x1 struct
Termination	1x1 struct
w	0.9000
wrongPre	[ ]
wrongSuf	[ ]
x0	1x100 double

Fig. 8. Output of STyLuS* for a case study with (i) $N = 100$ robots, (ii) TS with 1000 states, and (iii) an LTL formula corresponding to an NBA with 21 states.