The computation complexity of Nodetermine Büchi Automaton(NBA) has always been a problem for applications. So there are many effort attempt to simplify the NBA and do find some methods. One of the technique called *Partial Order Reduction* is to reduce the number of possible orderings that need to be analyzed for checking formule stated in a temporal logic as LTL. The main purpose is to reduce the state space of the transition system and there are some differece between Computer Science and Multi-agent system technically. The former use it to find a reduced system consists of a single path while the latter want to find all the Partial Order without loose much message.

There are many typos in the documents. Use a spell-checker in the editor.

Given a Linear Temporal Logic(LTL) formula, we can use the existing tools such as LTL2BA to construt the NBA $A_\phi = (\nu, \varepsilon)$ (which can also be defined as a tuple $\mathcal{B} = (Q_B, Q_B^0, \Sigma, \rightarrow_B, Q_B^F)$). Before consider the product with Transition System(TS) of Multi-agents, we only use the message within the NBA A_ϕ to get the sets of posets P_ϕ for simplification. It worth noting that P_ϕ is no longer a Graph and we only use it for the further computation with TS.

Add proper definition of $\mathcal{B}$ here, i.e., with all the elements property defined.

Use section...not bold. Follow a simple LaTeXtutorial.

A.Trimming the NBA

We only consider the *disjunctive normal form* (DNF) A_{DNF} of A_ϕ with the assumption that a optimal run in Buchi should not wasted cost at any node's self loop. Then, to trimming the useless edges, we defined the triangle property as definition 1.1. Once any three vertices v_1, v_2, v_3 in A_{DNF} satisfied the RT property, we can remove the edge ϵ_{13} while the feasibility of A_{DNF} would not be compromised. With a little abuse of notation, we call the NBA A^- after A_{DNF} been processed by RT property.

DNF of a NBA is equivalent to the original NBA. Namely, there is no simplification when using DNF.

Do not use explicit enumeration. Use environment and references.

Definition 1.1:(Redundance triangle property) Given three different vertices v_1, v_2, v_s in the A_{DNF}. . The function $L(\epsilon)$ means get the labels in the edge ϵ. we say that these three vertices satifsfy the RT property if:

(a) $\epsilon_{12} = v_1 \rightarrow v_2, \epsilon_{23} = v_2 \rightarrow v_3, \epsilon_{13} = v_2 \rightarrow v_3$ exist.

(b)$L(\epsilon_{12}) \wedge L(\epsilon_{12}) = L(\epsilon_{13})$

As we discussed last time: this is not correct by default. For general case, this would remove a lot of valid words. If you want to use it, then you need to show why it will not cause the loss of completeness in your algorithm.

B. Find all feasible path

A^- has the same vertices as A_ϕ while its edges is much less than that in A_ϕ. So find all feasible path in A^- can be much more simple in terms of time and computer internal storage. There many mature algorithms in finding all feasible path in a graph such as *Depth first Search*(DFS):

1) Choose a verties as the start point, and Build a Null list L_p.

2) Put the point into a path to a list.

3) Get the fisrt path in the path list and find all the unvisited neighbours of the last point in the path. Generate the new paths by adding the neighbours points. Remove the fatched path in the path list.

4) Go to step 3 until the path list get null.

5) Return the path list L_p.

You normally do not need to describe DFS. Also, the pseudo code above seems not correct. You need to use **DEPTH** to choose the next node to visit.

C. Calculate the poset

Parallel is a concept to describe the actions in the NBA so before finding the parallel relationship, we should change the states paths set $L_p]$ to the actions paths set L_a first. Given two action a_i, a_{i+1}, we use the *dependent* and *independent* to describe the parallel relationship as definition 1.2. So we can transform the L_a into a set of posets P with following algorithm:

L_p and L_a are not defined, which are important. After properly defining $\mathcal{B}$, you can define L_p and L_a here.

Same here. Do not use explicit enumeration. Use environment and references.

Definition 1.2:(Parallel relationship) Given an action path set L_a, for one path $l_a = [a_1, a_2, \cdots, a_n] \in L_a$. a_i and a_{i+1} $(a_i \neq a_{i+1})$ are *independent* $a_i || a_{i+1}$ (in l_a) if:There exist a path $l'_a \in L_a$ that $l'_a(j) = l_a(j), j \neq i, j \neq i+1$ $l'_a(i) = l_a(i+1), l'_a(i+1) = l_a(i)$. a_i and a_{i+1} are *dependent* if $l'_a \notin L_a$

Since you did not introduce FTS here, so probably "actions" should not be used. Instead, I would recommend to define this relation purely on the edges of $\mathcal{B}$. Namely, this relation should be stated w.r.t $\mathcal{B}$.

I think this relation should not be only about consecutive edges, namely two edges e_i and e_j can be parallel if they can appear in *any* order in a path (i.e., not just consecutive), right?

D. Build the poset graph Same here. Do not use explicit enumeration for algorithm number and line number. Use environment and references. The Poset p has two relationship as $||, <$. To be more visualized, we can change it to a graph $G = \nu, \epsilon$. ν is the nodes set containd by the action with order number defined in algorithm 1 line 10 and ϵ is the set of edges as the $<$ relation in p. Given an poset $p = 1 < 2, 1 < 4, 4 < 5, 2 < 3, 5 < 6, 5 < 7, 3 < 8, 6 < 8, 7 < 8$ with a map $Map = [1, 4, 5 :' rise', 2, 7, 8 :' put', 3, 6 :' goto']$, then we can get a graph:

All key variables in the pseudo code are normally defined in the main text. Please do so carefully.

How much is Alg.1 new? which part is from the book and what have you changed?

I would recommend to rewrite Alg.1 after you properly defined the variables.

Example.1

Given a LTL formula $\phi = <> (r1 \vee <> (r2 \vee <> (r3 \vee <> r4))) \vee <> (r5 \vee <> (r6 \vee <> (r7 \vee <> r8)))$, we can get a Büchi automatom A_ϕ with 44 states and

Algorithm 1: Generate poset set

input : actions path set L_a
output: set of Posets P

$L_{unvisited} = L_a$;
while $L_{unvisited} \neq Null$ **do**
  for l_a **in** L_a **do**
    Remove l_a in $P_{unvisited}$;
    $n = lenth(l_a)$;
    $G_{research} = \{l_a\}$;
    Poset $p_{l_a} = \{\}$;
    while $G_{research} \neq Null$ **do**
      Get the last one in $G_{research}$ and remove it;
      Build Search list $queue = \{[1:n]\}$ — [1:n] is the map for l_a;
      while $queue \neq \{\}$ **do**
        Fetch the last Map in $queue$;
        for i **in** $1 : n-1$ **do**
          Exchange the $Map(i)$ and $Map(i+1)$ and renamed as Map' ;
          l'_a =Sort l_a with order Map';
          if l'_a **in** L_a **then**
            add $(Map(i)||Map(i+1))$ to Poset p_{l_a}
          end
          if l'_a **not in** L_a **then**
            add $(Map(i) < Map(i+1))$ to Poset p_{l_a}
          end
          if l'_a **in** $L_{unvisited}$ **then**
            Remove l'_a from $L_{unvisited}$;
            add Map' to $queue$;
          end
        end
      end
    end
    add Poset p_{l_a} into Poset list P
  end
end

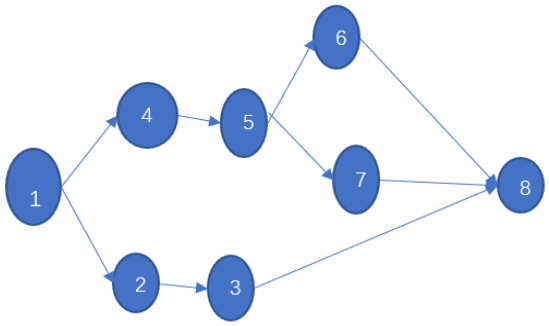

Figure 1: Poset graph

439 edges which is too complexity to be drawed into a graph. After trimming the A_ϕ, we get a much simplified graph A^- as fig.2 showed. And we can find 283 feasible path from initial node to the accept states. Then, using the algorithm.1, we can get the posets which shows the detail very intuitional as the table.1 showed. In the end, we can rebuild a graph of poset to show the relationships between the actions as fig.3 showed.

"Eventually" is written as $\diamond$, "and" and "or" are $\wedge$ and $\vee$.

As discussed last time, can you improve the alg. to achieve the following:

- find a path $\mathbf{p} = e_1 e_2 \cdots e_N$ in NBA during DFS.
- store it in $\mathcal{P} = \{\mathbf{p}\}$.
- use other already-found paths in $\mathcal{P}$ to determine the poset *specifically* for $\mathbf{p}$.

This would be an *anytime* algorithm, which is asymptotically complete as time goes to infinity. The benefit is that you do not need to find all simple paths in NBA in one shot.

Remark.1

The reason why make a map from a actions path l_a to a sequence from 1 to n is to avoid the interference with the same action. Once the actions α and β existed in the l_a more than twice, we may found that the $\alpha||\beta$ while $\alpha < \beta$ at the same time.

What is this map? your explanation sounds like a direct enumeration.

To be proved

We need to proved that, once get a feasible path in L_a, can find all other path stand for the same . for any path

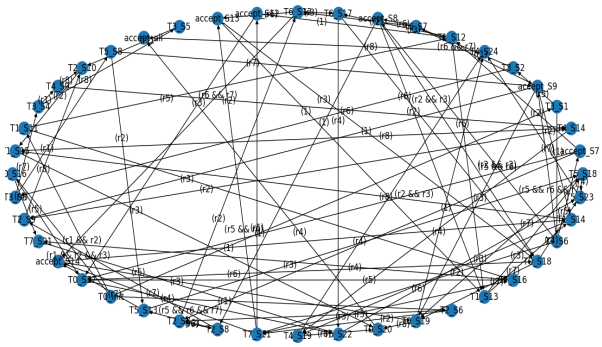

Figure 2: Trimmed NBA graph

Table 1: Poset

parallel	(0,1)(0,5),(0,6),(0,7),(1,2),(1,3),(1,4),(2,5),(2,6),(2,7),(3,5),(3,6),(3,7),(4,5),(4,6),(4,7)
less_than	(0, 2), (1, 5), (2, 3), (3, 4), (5, 6), (6, 7)
action_map	[’(r1)’,’(r5)’,’(r2)’,’(r3)’,’(r4)’,’(r6)’,’(r7)’,’(r8)’]

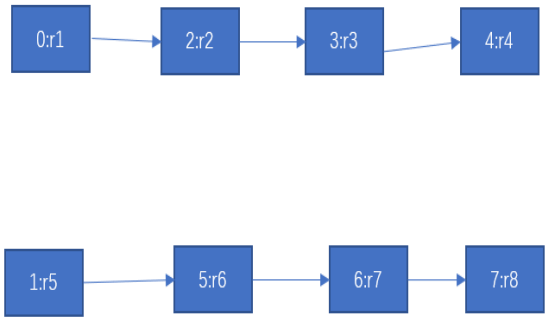

Figure 3: Poset graph