**Supplementary file of “Livenees Analysis and Deadlock Controller Design for Flexible Assembly Systems Using Petri Nets”**

1. Basic definitions and concepts about Petri nets

A *generalized* Petri net is a four-tuple *N* = (*P*, *T*, *F*, *W*), where *P* and *T* are the sets of places and transitions, respectively. *F* ⊆ (*P* × *T*) ∪ (*T* × *P*) is the set of directed arcs. *W*: *F* → *Z* \ {0} assigns the weights to arcs, where *Z* ≡ {0, 1, 2, …}. If ∀*f* ∈ *W*, *W*(*f*) = 1, *N* is called an *ordinary* net and denoted as *N* = (*P*, *T*, *F*) for short.

Let *x* ∈ *P* ∪ *T*. The preset and post sets of *x* is •*x* = {*y* ∈ *P* ∪ *T* | (*y*, *x*) ∈ *F*} and *x*• = {*y* ∈ *P* ∪ *T* | (*x*, *y*) ∈ *F*}, respectively. For *X* ⊆ *P* ∪ *T*, let •*X* = ∪*x*∈*X*•*x* and *X*• = ∪*x*∈*X x*•. The *incidence matrix* of *N* denoted as [*N*], is a |*P*| × |*T*| matrix satisfying [*Nij*] = [*Nij*]+ − [*Nij*]−, where [*Nij*]+ = *W*(*tj*, *pi*) and [*Nij*]− = *W*(*pi*, *tj*).

A marking or state of *N* is a mapping *M*: *P* → *Z*. Given a place *p* ∈ *P* and a marking *M*, *M*(*p*) denotes the number of tokens in *p* at *M*. Let *S* ⊆ *P* be a set of places. *M*(*S*) denotes the sum of tokens in all places of *S* at *M*, i.e., *M*(*S*) = Σ*p*∈*SM*(*p*). Let (*N*, *M*0) denote the Petri net *N* with initial marking *M*0.

Let *Zk =* {1, 2, …, *k*}, *k* ∈ *Z* \{0}. A transition *t* ∈ *T* is *enabled* at a marking *M* if ∀*p* ∈ •*t*, *M*(*p*) ≥ *W*(*p*, *t*), denoted by *M*[*t*>. An enabled transition *t* can fire at *M*, resulting in a new making *M*1, denoted by *M* [*t* > *M*1, where *M*1(*p*) = *M*(*p*) − *W*(*p*, *t*) + *W*(*t*, *p*). A sequence of transitions δ = *t*1*t*2*…tk*, is *friable* from *M*, if *Mi*[*ti* > *Mi+*1, *i* ∈ *Zk*, hold, where *M*1 = *M* and *Mi* is called a reachable marking from *M*. Let *R*(*N*, *M*0) denote the set of all reachable markings from *M*0. Furthermore, *M* = *M*0 + [*N*]*Y* is the *state equation* of (*N*, *M*0), where *Y* is a vector whose *i*th entry denotes the number of occurrences of *ti* in δ. For economy of space, ∑*p*∈*PM*(*p*)*p* is used to denote a marking *M*. For example, marking *M* = (*M*(*p*1), *M*(*p*2), *M*(*p*3)) = (1, 2, 0) can be denoted by *M* = *p*1 + 2*p*2.

A transition *t* ∈ *T* is *live* if ∀*M* ∈ *R*(*N*, *M*0), ∃*M′* ∈ *R*(*N*, *M*) ∍ *M′* [*t* >, and it is *dead* at *M* ∈ *R*(*N*, *M*0) if ∃*M′* ∈ *R*(*N*, *M*) ∍ *M′* [*t* >. A marking *M* ∈ *R*(*N*, *M*0) is a (partial) *deadlock* if ∃*t* ∈ *T* ∍ *t* is dead at *M*. A Petri net (*N*, *M*0) is live if every transition is live.

Given a Petri net *N* = (*P*, *T*, *F*, *W*), a *path* in *N* is a string η = *x*1*x*2 *… xk*, *xi­* ∈ *T* ∪ *P*, (*xi*, *xi+*1) ∈ *F*, *i* ∈ *Zk−*1, and path η is *elementary* if all its nodes are different (perhaps) except *x*1 and *xk*. Further, path γ = *x*1*x*2*…xk* is a *circuit* if *x*1 = *xk*. The reverse of *N* is a Petri net *N*−1 = (*P*, *T*, *F*−1, *W*−1), where *F*−1 = {(*x*, *y*) | (*y*, *x*) ∈ *F*} and *W*−1(*x*, *y*) = *W*(*y*, *x*) for (*x*, *y*) ∈ *F*−1. In other words, reversing all arcs of a net generates its reverse net.

Two marked Petri nets, (*Ni*, *Mi*0) = (*Pi*, *Ti*, *Fi*, *Wi*, *Mi*0), *i* ∈ *Z*2, are *compatible* if ∀*p* ∈ *P*1 ∩ *P*2, *M*10(*p*) = *M*20(*p*). The composition of two compatible marked Petri nets (*N*1, *M*10) and (*N*2, *M*20) is a marked Petri net (*N*1, *M*10) ⊗ (*N*2, *M*20) = (*P*, *T*, *F*, *W*, *M*0), where *P* = *P*1 ∪ *P*2, *T* = *T*1 ∪ *T*2, *F* = *F*1 ∪ *F*2, *W* = *W*1 ∪ *W*2, ∀*p*1∈ *P*1, ∀*p*2 ∈ *P*2, *M*0(*p*1) = *M*10(*p*1), and *M*0(*p*2) = *M*20(*p*2).

2. Algorithms to compute all A-circuit and closed Ω-structures in APNMR

Algorithm 1 (compute all A-paths and A-chains)

Input: APNMR (*N*, *M*0);

Output: Ψ, the set of all A-paths, and *K*, the set of all A-chains;

1. Let τ = {α = *pt* | *p* ∈ *PA* and *t* = *p*• ∩ *T*} and Ψ = τ;
2. **for each** α = *pt* ∈ τ
3. **for each** α′ = *p*1′*t*1′…*pn*′*tn*′ ∈ Ψ
4. **if** *t* ∈ •*p*1′
5. Let α′′ = *pt p*1′*t*1′…*pn*′*tn*′ and add α′′ into Ψ;
6. **end**
7. **if** *tn*′ ∈•*p*
8. Let α′′ = *p*1′*t*1′…*pn*′*tn*′*pt* and add α′′ into Ψ;
9. **end**
10. **end**
11. **end**
12. Let Κ = Ψ;
13. **for each** α = *p*1*t*1…*pktk* ∈ Ψ //Ψ is the set of all A-paths
14. **for each** α′ = *p*1′*t*1′…*pn*′*tn*′ **∈** Κ
15. **if** (*r*)*tk* = ℜ(*p*1′)
16. Let α1 = αα′ = *p*1*t*1…*pktkp*1′*t*1′…*pn*′*tn*′ and add α1 into Κ;
17. **end**
18. **end**
19. **end**
20. output Ψ and Κ;

*The computational complexity of Algorithm 1*: let *n* be the number of activity places, and hence |τ| = *n*. Note that there are at most A-paths in the system, that is, |Ψ| ≤ . Thus, the complexity of lines 2−11 is *O*(). On the other hand, since there are at most A-chains, the complexity of lines 13−19 is *O*(()()). Therefore, the complexity of Algorithm 1 can be considered as *O*().

Algorithm 2 (compute all A-circuits)

Input: APNMR (*N*, *M*0);

Output: Ξ, the set of all A-circuits;

1. According to Algorithm 1, obtain K, the set of all A-chains;
2. let Ξ = Κ;
3. **for each** θ = *p*1*t*1…*pktk* ∈ Κ
4. **if** (*r*)*tk ≠* ℜ(*p*1)
5. Delete θ from Ξ;
6. **end**
7. **end**
8. output Ξ;

*The computational complexity of Algorithm 2*: The analysis shows that there are most A-chains, hence, the complexity of Algorithm 2 is *O*(), where *n* denotes the number of activity places.

Algorithm 3 (compute all closed Ω-structures)

Input: APNMR (*N*, *M*0);

Output: Ξ, the set of all A-circuits;

1. According to Algorithm 1, obtain Ψ (the set of all A-paths) and *K* (the set of all A-chains);
2. Let Ε = ∅;
3. **for each** A-chainα = *p*1*t*1…*pktk*∈ Κ
4. **for each** A-path α′ = *p*1′*t*1′…*pn*′*tn*′ ∈ Ψ
5. **if** ℘(α) ∩ ℘(α′) = ∅ and *tk* = *tn*′
6. let *v* = (α, α′) and add *v* into Ε;
7. **end**
8. **end**
9. **end**
10. let Τ = Ε;
11. **for each** *v-*structure *v* = (α, α′) ∈ Ε
12. **for each** Ω-structure *w* = α1α1′α2α2′…α*k*α*k*′ ∈ Τ
13. **if** ℜ(ϑ(α′)) = ℜ(ϑ(α1))
14. Let *w′* = αα′α1α1′α2α2′…α*k*α*k*′ and add *w′* into Τ;
15. **end**
16. **if** ℜ(ϑ(α*k*′)) = ℜ(ϑ(α))
17. Let *w′* = α1α1′α2α2′…α*k*α*k*′αα′ and add *w′* into Τ;
18. **end**
19. **end**
20. let Θ = ∅;
21. **for each** *w*=α1α1′α2α2′…α*k*α*k*′=(Π1, Π2)∈Τ //Π1 ={α*i*, *i*∈*Zk*}, Π2 = {α*i*′,*i*∈*Zk*}
22. **if** ℘(Π1) ∩ ℘(Π2) = ∅ and ℜ(ϑ(α1)) = ℜ(ϑ(α*k*′))
23. Add *w* into Θ;
24. **end**
25. **end**
26. output Θ;

*The computational complexity of Algorithm 3*: since the number of A-paths (resp. A-chains) is less than (resp. ), then the number of *v-structures* is less than ][] and the complexity of lines 3−9 is *O*(][]). Each Ω-structure is constructed by several *v*-structures, thus there are at most Ω-structures. Therefore, the complexity of Algorithm 3 is *O*().