Individual requirements

Pengyi Li(UPI:pli552)

October 10, 2014

1 English requirements(Functional requirements)

For Line conveyor one(module one)

Actors:operator(who will put bottles to module one),module two(round conveyor before filling water),sensor one(check module one's state),sensor two(check module two's state)

For Round conveyor before filling water(module two)

Actors: module one(Line conveyor one),module three (Round conveyor fill water),sensor two(check module two's state),sensor three (check module three's state)

1. Operator always gets bottles from the factory storage.
2. Operator always records the bottles' number that will be put to module one.
3. Sensor one always checks module one's state.
4. Module one has three states: not active, active, having one bottle approaches the module two.
5. Operator can put a bottle to module one when module one is "active" or "having one bottle approaches the module two".
6. Operator can not do anything until model is active or having one bottle approaches the module two.
7. Sensor two always checks module two's state.
8. Module two just has two states : active idle(no bottle in it) and active full(having bottle in it)
9. If module one has one bottle approaches the module two, it eventually puts a bottle to module two.
10. Module one eventually puts a bottle to module two when module two's state is "active idle".

11. If module two state is “active idle”, it eventually gets a bottle from module one.

12. Module two eventually gets a bottle from module one,when module one has one bottle approaches the module two.

13. Module two state “active full” eventually changes to “active idle”.

14. Module two state “active idle ”eventually changes to “active full”.

15. Module three has three states: empty,bottle without water,bottle with water.

16. If module three state is “empty”or “bottle with water”, it eventually gets a bottle from module two.

17. Module three eventually gets a bottle from module two, when module two state is “active full”.

18. If Module two state is “active full”, it eventually puts a bottle to module three.

19. Module two eventually puts a bottle,when module three state is “empty”or “bottle with water”,.

2 CTL requirements

Boilerplates:
BP1:The <system> always reaches <state-label>. system |=AGAF state-label

1. AGAF Operator.Getbottles

2. AGAF Operator.Recordnumber

3. AGAF Operator.Checkstate

BP2: Every <request> of the <system> is eventually reaches <state-label>. system |= AG(request => AF state-label)

1. AG (([ModuleOne.state=active] ∨ [ModuleOne.state=approach]) => AF Moduleone.Getbottles)

2. AG (ModuleOne.state=notactive => AF ¬(Moduleone.Getbottles∧Moduleone.Putbottles)

3. AG (ModuleOne.state=approach => AF ModuleOne.Putbottles)

4. AG (ModuleTwo.state=activeidle => AF ModuleOne.Putbottles)

5. AG (ModuleOne.state=approach => AF ModuleTwo.Getbottles)

6. AG (ModuleTwo.state=activeidle => AF ModuleTwo.Getbottles)
7. AG (ModuleTwo.state=activefull => AF ModuleTwo.state=activeidle)
8. AG (ModuleTwo.state=activeidle => AF ModuleTwo.state=activefull)
9. AG ((ModuleThree.state=empty∨ ModuleThree.state=bottlewithwater) => AF ModuleThree.Getbottles)
10. AG (ModuleTwo.state=activefull =>AF ModuleThree.Getbottles)
11. AG ((ModuleThree.state=empty∨ ModuleThree.state=bottlewithwater) => AF ModuleTwo.Putbottles)
12. AG (ModuleTwo.state=activefull =>AF ModuleTwo.Putbottles)

BP3: The <system> always be <state-label>. system |=AG state-label

1. AG ([ModuleOne.state=notactive] ∨[ModuleOne.state=active] ∨ [ModuleOne.state=approach])
2. AG ([ModuleTwo.state=activeidle] ∨[ModuleTwo.state=activefull])
3. AG ([ModuleThree.state=empty]∨ [ModuleThree.state=bottlewithoutwater]∨[ModuleThree.state=bottlewithwater])
4. AG SensorOne.Check
5. AG SensorTwo.Check
6. AG SensorThree.Check

BP4: Every <request> of the <system> is next reaches <state-label>. system |= AG (request => AX state-label)

1. AG ((ModuleOne.state=approach ∧ModuleTwo.state=activeidle) => AX Moduleone.Putbottles)

3 Non-functional requirements

1. Module one is deadlock free.
2. Module two is deadlock free.
3. Operator can put at least 1000000 bottles for module one.
4. Module one can be used for most industry system.
5. Sensor failure rate $\in[0.00000001, 0.0000001]$
6. If module one is broke, it will recover in 6 mins.
7. If module one is broke, it will recover in 6 mins.

4 verifier requirements

1. A[]A<> Operator.Getbottles
2. A[]A<> Operator.Recordnumber
3. A[]A<> Operator.Checkstate
4. A[] (([ModuleOne.state=active] || [ModuleOne.state=approach]) imply A<> Moduleone.Getbottles)
5. A[] (ModuleOne.state=notactive imply A<>not (Moduleone.Getbottles && Moduleone.Putbottles)
6. A[] (ModuleOne.state=approach imply A<> ModuleOne.Putbottles)
7. A[] (ModuleTwo.state=activeidle imply A<> ModuleOne.Putbottles)
8. A[] (ModuleOne.state=approach implyA<> ModuleTwo.Getbottles)
9. A[] (ModuleTwo.state=activeidle imply A<> ModuleTwo.Getbottles)
10. A[] (ModuleTwo.state=activefull imply A<> ModuleTwo.state=activeidle)
11. A[] (ModuleTwo.state=activeidle imply A<> ModuleTwo.state=activefull)
12. A[] ((ModuleThree.state=empty || ModuleThree.state=bottlewithwater) imply A<>ModuleThree.Getbottles)
13. A[] (ModuleTwo.state=activefull imply A<> ModuleThree.Getbottles)
14. A[] ((ModuleThree.state=empty || ModuleThree.state=bottlewithwater) imply A<> ModuleTwo.Putbottles)
15. A[] (ModuleTwo.state=activefull imply A<> ModuleTwo.Putbottles)
16. A[] ([ModuleOne.state=notactive] || [ModuleOne.state=active] || [ModuleOne.state=approach])
17. A[] ([ModuleTwo.state=activeidle] ||[ModuleTwo.state=activefull])
18. A[] ([ModuleThree.state=empty] || [ModuleThree.state=bottlewithoutwater] || [ModuleThree.state=bottlewithwater])
19. A[] SensorOne.Check
20. A[] SensorTwo.Check
21. A[] SensorThree.Check
22. A[] not deadlock