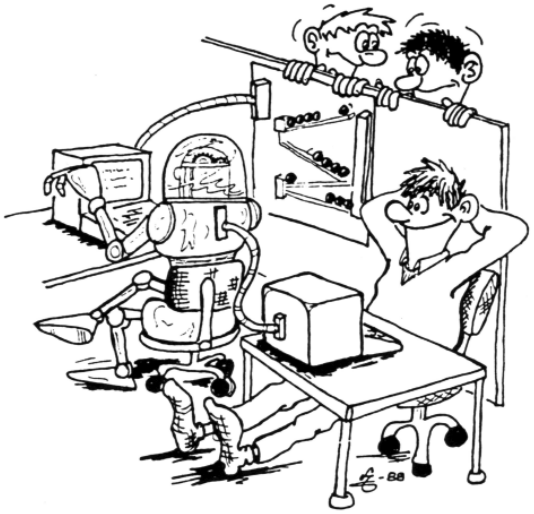
Industriautomation

Tentamen SSY 066, fredag 17/01–2020, fm

Lärare: Kristofer Bengtsson, 0768-979561



Fullständig lösning ska lämnas på samtliga uppgifter. I förekommande fall av tvetydigt formulerade tentamensuppgifter ska den föreslagna lösningen och eventuella antaganden motiveras. Examinator förbehåller sig rätten att godkänna rimligheten i antaganden och motiveringar.

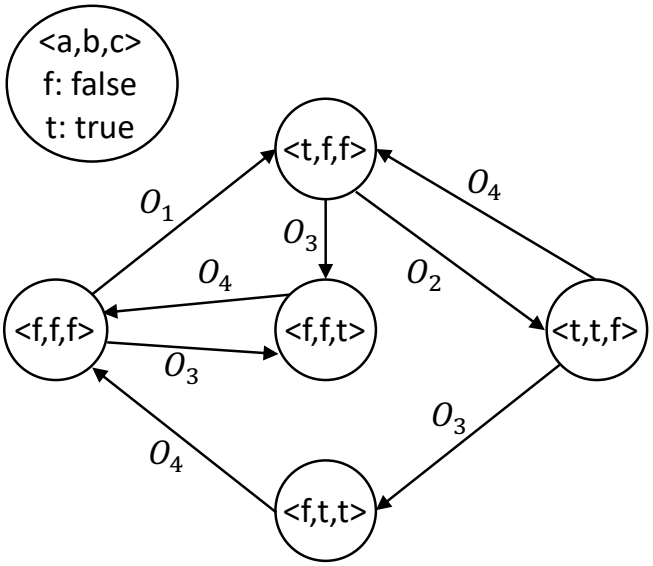
Totalt omfattar tentamen 40 poäng. För betygen tre, fyra och fem krävs 16, 24 resp. 32 poäng. Lösningar anslås direkt efter tentatillfället på kursens hemsida i Pingpong. Granskning av rättningen sker fredag den 7 februari kl. 12:30 – 13:30 på institutionen (EDIT vån 5).

OBS. Inga hjälpmedel är tillåtna.

Uppgift 1

I denna uppgift skall du skriva ner de guards och actions som de fyra operationerna, O_1, O_2, O_3, O_4 har, så att dess exekvering beskrivs av grafen. Det finns tre boolska variabler, a, b, och c och deras värden beskrivs i varje tillstånd i grafen med notationen <a, b, c>. Tex om alla variabler är false står det <f, f, f> och om tex bara b är sann, blir det <f, t, f>

Du skall försöka hitta det minsta möjliga guard-uttrycket för operationerna.



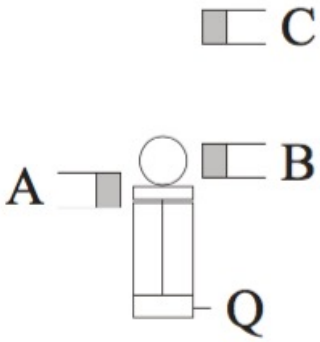
(6 poäng)

Lösning uppgift 1:

O_1: $\neg a \wedge \neg c$ / $a \coloneqq true$
O_2: $a \wedge \neg b$ / $b \coloneqq true$
O_3: $\neg c$ / $a \coloneqq false; c \coloneqq true$
O_4: $b \vee c$ / $b \coloneqq false; c \coloneqq false$

Uppgift 2

I denna uppgift skall du styra en liten cylinder som kan åka mellan två lägen, nere och uppe. När cylindern är nere är givare A sann. För att den skall åka upp, skall utsignalen Q sättas sann, då åker den upp till sitt ”uppläge”. För att cylindern skall åka ner, skall Q sättas till falsk.



Cylindern kan lyfta kulor. Om en kula finns på plats i det nedre läget är sensor B sann, och om en kula finns på plats när cylindern är uppe, blir sensor C sann.

Din uppgift är att implementera följande beteende med hjälp av ladderkod:

När cylindern är nere och det läggs en kula på plats (A och B är sanna) skall din kod

1. först lyfta upp kulan till sensor C.

2. Sedan skall cylindern omedelbart åka ner igen till sensor A och B med kulan.

3. Därefter skall cylindern direkt åka upp igen.

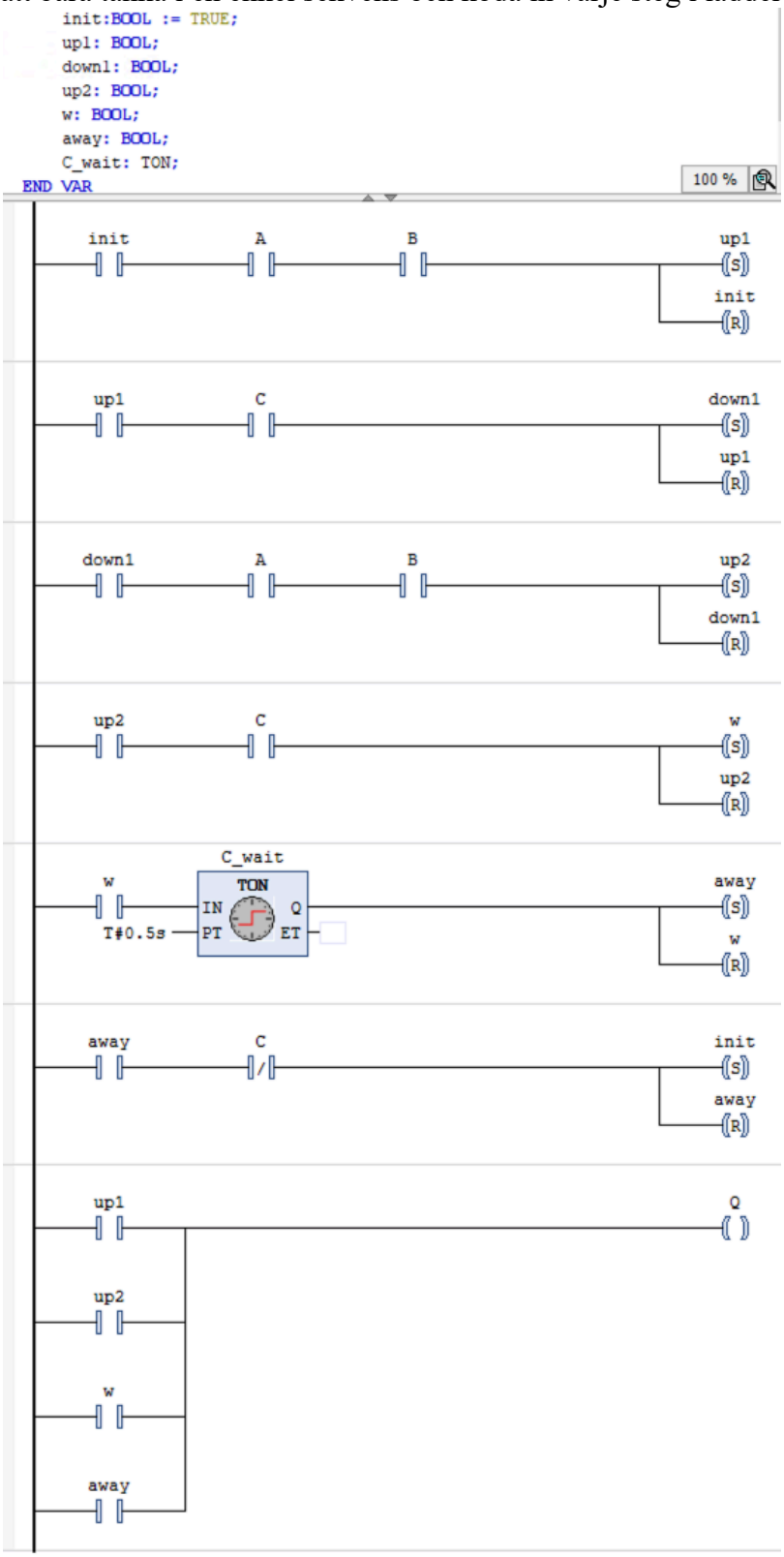
4. När kulan stannar vid uppläget den andra gången skall cylindern stå still under 0.5 sekunder eftersom kulan kan skaka lite och sensor C kanske kan blinka till.

5. Efter denna väntetid skall din kod vänta på att kulan plockas bort av någon annan (C blir falsk) och skall då köra ner cylindern. När den är nere skall koden vara redo att köra en ny kula.

(6 poäng)

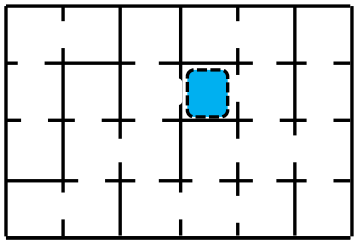
Möjlig lösning uppgift 2

Det enklaste är att bara tänka i en enkel sekvens och koda in varje steg i ladder enligt:



Uppgift 3

Nu skall vi utveckla en robot som kan lära sig att hitta i olika ”hus”. Vi har en liten blå robot (vi kallar den B) som kan flytta sig från rum till rum i en 2D-värld, genom att åka upp (U), ner (N), höger (H) eller vänster (V). Roboten kan endast utföra en operation om det finns en dörr i respektive riktning och varje operation tar roboten till rummet bredvid.

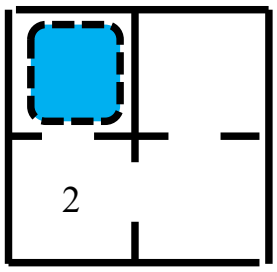


B släpps ner i olika hus på en slumpmässig position. Robotens första uppgift är då att kartlägga våningen genom att köra runt lite slumpmässigt och kontinuerligt uppdatera sin modell, som roboten sedan kommer att använda i sin grafsökningsalgoritm för att hitta snabbaste vägen mellan olika rum. Modellen som roboten uppdaterar består av de fyra operationer (U, N, H, V) med guards och actions och ett antal variabler.

Din uppgift är att beskriva hur roboten uppdaterar sin modell så att den, när den besökt alla rum, kan använda modellen i sin grafsökningsalgoritm för att hitta kortaste vägen mellan två rum.

a) Din första uppgift är att införa variabler så att B kan hålla ordning på i vilket rum den befinner sig. Roboten kommer att åka runt i huset med hjälp av sina operationer. Skriv eventuellt nya guards och actions för operationerna så att B hela tiden vet var den är. Observera att du inte vet något om huset när B börjar, utan B startar i något slumpmässigt rum och navigerar utifrån det. Du behöver inte tänka på själva styrningen av roboten, utan när en operation sker, löser roboten hur den tar sig till nästa rum. En operation sker endast om det finns en dörr i den riktningen. I modellen behöver inte en operation ta någon tid.

b) Nu när B vet var den är, kan den bygga upp sin modell var den kan och inte kan åka. Fundera ut ett sätt hur B kan uppdatera operationernas guards och actions så att modellen kan användas vid optimering. B ser i varje rum i vilken riktning det går att åka. Du behöver inte programmera något, utan skriv ner hur dina operationer har uppdaterats när den kommer till rum 2 i det lilla huset till höger där du startar som på bilden och skriv också de guards och actions för operationerna som gäller alla fyra rummen.



c) Nu har B en bra modell för hur den kan åka runt i huset och givet sin nuvarande position och en mål-position kan den använda en optimeringsalgoritm för att hitta kortaste vägen.
- Om huset är stort kan det dock ta lång tid att söka igenom med B:s lite halvdåliga graf-sökningsalgoritm. Så B behöver ett lämpligt sätt att sluta söka i vissa av grafens grenar. Vilket skulle vara ett lämpligt villkor på ett graftillstånd för att inte fortsätta sökningen från det tillståndet?
- Roboten vill att direkt när målet hittas, skall det vara den kortaste vägen. Vilken algoritm som ni lärt er i kursen kan garantera det?

(8 poäng)

Möjlig lösning uppgift 3:

a) Där roboten startar blir origo och vart beräkningen utgår ifrån. Jag håller koll på positionen med variablerna x (där jag tänker att x räknar +1 åt höger och -1 år vänster) och y (räknar +1 ner och -1 upp). Init: x= 0, y=0

Vi behöver inga nya guards, men lägger till följande actions:
$U: y \coloneqq y - 1$
$D: y \coloneqq y + 1$
$R: x \coloneqq x + 1$
$L: x \coloneqq x - 1$

b) I varje rum som roboten kommer till, lägger den till nya guards för där det går att köra i respektive riktning.

I rum 2 befinner sig roboten enligt mitt sätt att definiera a, i x=0, y=1. Tillsammans med de guards som den fick i startrummet och de i rum 2 blir:

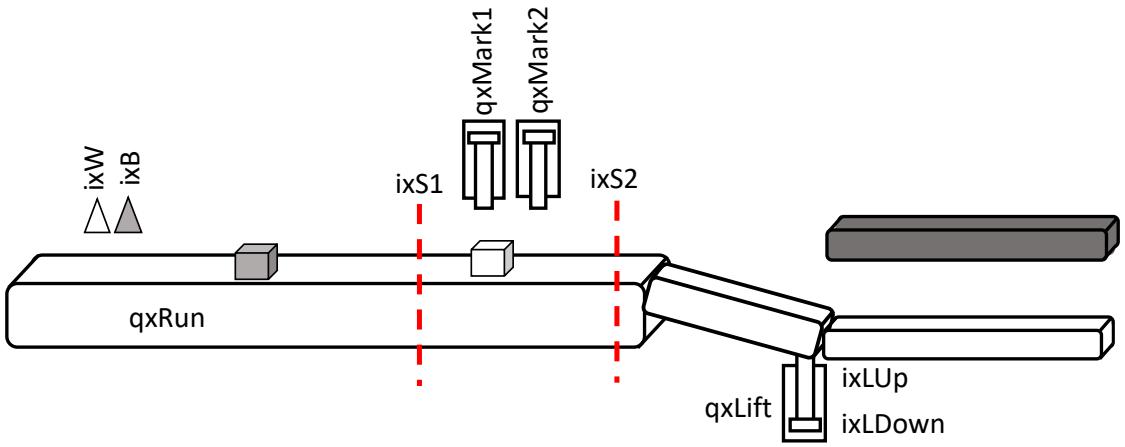
$U: x = 0 \wedge y = 1 \;/\; y \coloneqq y - 1$
$D: x = 0 \wedge y = 0 \;/\; y \coloneqq y + 1$
$R: x = 0 \wedge y = 1 \;/\; x \coloneqq x + 1$
$L: \qquad\qquad\quad /\; x \coloneqq x - 1$

För all fyra rummen blir det (med förenklingar):
$U: y = 1 \wedge (x = 0 \vee x = 1) \;/\; y \coloneqq y - 1$
$D: y = 0 \wedge (x = 0 \vee x = 1) \;/\; y \coloneqq y + 1$
$R: x = 0 \wedge y = 1 \;/\; x \coloneqq x + 1$
$L: x = 1 \wedge y = 1 \;/\; x \coloneqq x - 1$

c) Eftersom roboten kan åka fram och tillbaka mellan olika rum så kommer grafen som vi söker i bli riktigt stor. Men det kortaste vägen mellan två rum kommer aldrig kräva att vi besöker samma rum två gånger. Därför är det lämpligt att avbryta sökningen om vi kommer till samma rum som vi redan besökt fast via en annan väg.

Breadth-first search garanterar att vi hittar den kortaste vägen första gången vi hittar målet.

Uppgift 4



Nu är det dags för lite SFC-styrning! Maskinen ovan är en märk- och sorterings-maskin där vita och svarta lådor susar fram på transportbandet och märks med en av de två märk-maskinerna och sorteras sedan med liften, svarta där uppe och vita där nere. I denna uppgift skall du implementera styrningen av maskinen med hjälp av SFC.

- Transportbandet styrs med qxRun (vid true så kör den) och skall nästan aldrig vara avstängd. Märkningen hinner maskinen nämligen göra i farten, men tyvärr är liften, som styrs med qxLift (vid true så åker den upp och false så åker den ner), lite långsam. Om inte liften är i rätt position när en kub passerar lasersensorn ixS2, måste transportbandet stanna tills liften är i rätt position. Positionen indikeras med sensorerna ixLDown (den är nere och kan ta emot vita) och ixLUp (den är uppe och kan ta emot svarta).

- Sensorerna ixW och ixB i början av maskinen mäter lådornas färg. Om det är en vit kub som åker förbi sensorerna, blir ixW true en kort stund, och om det är en svart låda blir ixB true en kort stund.

- När en låda passerar lasersensor ixS1, och en annan låda inte har hunnit passera ixW eller ixB, skall märkningsmaskin 1 användas (som styrs med qxMark1, där den skall vara true endast en scancykel för att märka lådan). Det tar 0.5 sekunder för en låda att åka från ixS1 till maskin 1. Om en annan låda redan har hunnit passera ixW eller ixB när lådan är vi ixS1, skall maskin två användas istället (qxMark2). Det tar 0.8 sekunder för en låda att åka från ixS1 till maskin 2.

- Lådorna kommer åkande med ett varierande avstånd mellan sig, men lådan som passerar ixS1 kommer alltid hinna passera liften innan nästa kommer dit, och det kan max finnas två lådor mellan ixW / ixB och ixS1.

Om du vill använda flera SFC:er kan du införa egna variabler som kan skrivas och läsas från alla SFC:er. Observera: I denna uppgift får du inte använda komplicerad logik med if-satser skriven i strukturerad text! Du får använda vanliga action-rutor och även använda enklare tilldelningar skrivet i ST i ett steg, typ:

N | a := 1; foo[i] := bar[j]; kalle := true; x := y + 1;

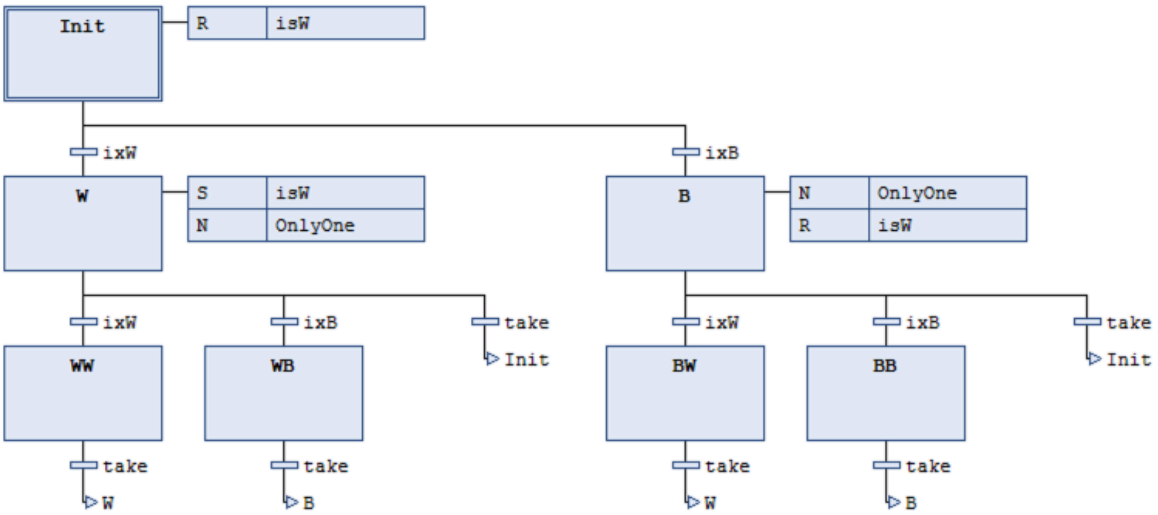
(10 poäng)

Möjlig lösning uppgift 4

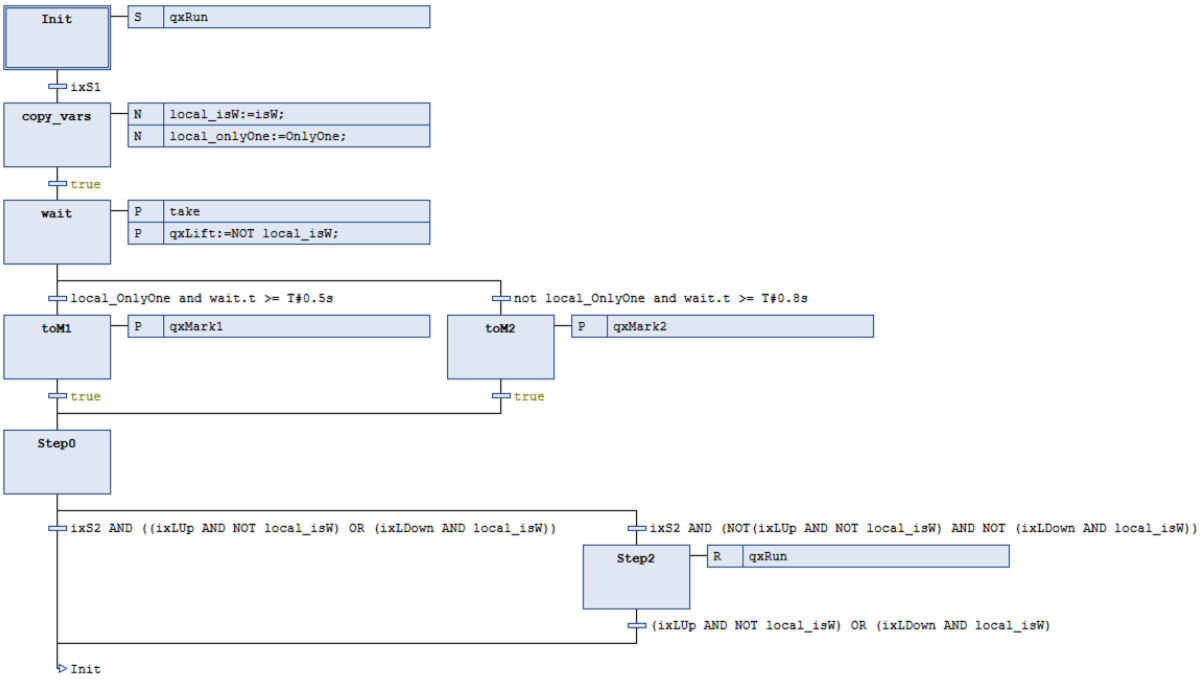
För att kunna hantera flera kuber mellan mätningen och sensor 1 behöver vi göra en liten kö. Jag inför följande variabler för kön:

```
OnlyOne: BOOL; // True if only one cube is in the que, and not two
isW: BOOL;     // The cube first in the que is white
take: BOOL;    // takes one cube from the que
```

Följande SFC-kö hanterar två kuber:

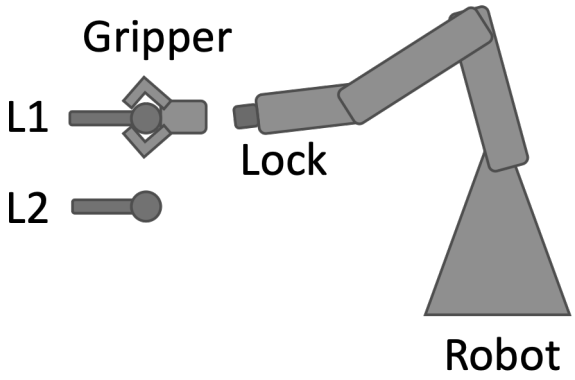


Sedan styr vi märkningen och sorteringen med följande SFC:



Uppgift 5

Det finns många situationer som är lite kluriga att hantera med ett styrsystem. I denna uppgift skall vi titta på när en robot hämtar och lämnar ett grip-verktyg på två olika positioner.



Vi har tre resurser i vårt lilla system, en robot, ett gripverktyg (gripper) och en låsmekanisk (lock) som kopplar ihop roboten med grippern.

Grippern styrs med utsignalen qxOpen (öppnar när den är sann och stänger när den är falsk) och har sensorerna ixOpened och ixClosed. Låset styrs med qxLock (låser vid sann, öppnar vid falsk) med sensorn ixLocked. Roboten styrs med utsignalerna qxToL1, qxToL2 och qxToHome (roboten åker till positionen vid sann) med sensorerna ixAtL1, ixAtL2 och ixAtHome (som är sanna när roboten är i den positionen). Roboten kan åka mellan alla positioner.

Systemet har följande specifikationer:

1. Endast en av robotens qxToL1, qxToL2 och qxToHome får vara sann samtidigt
2. Lock får endast låsa om roboten är i samma position som grippern
3. Lock får endast låsa upp (släppa grippern) om den är antingen i position L1 eller L2 och om grippern är stängd
4. Om roboten bär på grippern får den endast åka till L1 eller L2 om grippern är öppen.
5. Roboten får åka ifrån L1 eller L2 om den inte har grippern eller om den har grippern, måste grippern vara öppen.
6. Grippern får endast vara öppen om den sitter på roboten.

a) Beskriv guards och actions för operationerna Open, Close, Lock, Unlock, ToL1, ToL2 och ToHome så att det inte går att bryta mot specifikationen ovan men att en operation kan köra i alla tillfällen när det inte bryter mot dem. Operationerna skall också sätta lämpliga utgångar. Observera att operationerna tar tid och har både pre- och postconditions. Du behöver troligen även några interna variabler och någon operation kanske behöver finnas i flera versioner.

b) Nu skall du implementera ett ladderprogram som kan lämna grippern på L2, men som skall fungera oavsett i vilket position roboten, låset och grippern finns när knappen trycks in. Användaren trycker på knappen, ixGToL2, som sedan kommer att vara sann fram tills grippern är på L2. Du behöver inte implementera operationerna från a utan kan köra dem med signalerna {op_namn}.run (tex close.run stänger grippern). Observera att dina precondition gäller som i a, vilket innebär att en operations precondition och runsignalen måste vara sann för att den skall starta. Det finns inga andra operationssignaler utan du vet om de är färdiga genom att titta på de variabler den sätter.

   Observera att om grippern sitter på L1 måste den hämtas först.

(10 poäng)

Möjlig lösning uppgift 5

a) För att hålla iordning på var grippern är och om roboten är upptagen inför jag följande variabler:
Grippern har boolska gOnL1, gOnL1 och gOnR, där gOnL1=true som init
r : bool. Init r = F

Jag använder även T = true och F = false i denna uppgift och ↑ är precondition och ↓ är postcondition.

$Open^{\uparrow}: ixClosed \wedge gOnR \;/\; qxOpen \coloneqq T$
$Open^{\downarrow}: ixOpened$

$Close^{\uparrow}: ixOpened \wedge gOnR \wedge \neg r \;/\; qxOpen \coloneqq F$
$Close^{\downarrow}: ixClosed$

$Lock^{\uparrow}: \neg ixLocked \wedge ((gOnL1 \wedge ixAtL1) \vee (gOnL2 \wedge ixAtL2)) \;/\; qxLock \coloneqq T$
$Lock^{\downarrow}: ixLocked \;/\; gOnR \coloneqq T; gOnL1 \coloneqq F; gOnL2 \coloneqq F$

$UnLock\_L1^{\uparrow}: ixLocked \wedge ixClosed \wedge (gOnR \wedge ixAtL1) \;/\; qxLock \coloneqq F$
$UnLock\_L1^{\downarrow}: \neg ixLocked \;/\; gOnL1 \coloneqq T; gOnR \coloneqq F$

$UnLock\_L2^{\uparrow}: ixLocked \wedge ixClosed \wedge (gOnR \wedge ixAtL2) \;/\; qxLock \coloneqq F$
$UnLock\_L2^{\downarrow}: \neg ixLocked \;/\; gOnL2 \coloneqq T; gOnR \coloneqq F$

$ToL1^{\uparrow}: \neg ixAtL1 \wedge \neg r \wedge (ixOpened \vee \neg gOnR) \;/\; qxToL1 \coloneqq T; r \coloneqq T$
$ToL1^{\downarrow}: ixAtL1 \;/\; \mathrm{r} \coloneqq F;\; qxToL1 \coloneqq F$

$ToL2^{\uparrow}: \neg ixAtL2 \wedge \neg r \wedge (ixOpened \vee \neg gOnR) \;/\; qxToL2 \coloneqq T; r \coloneqq T$
$ToL2^{\downarrow}: ixAtL2 \;/\; \mathrm{r} \coloneqq F;\; qxToL2 \coloneqq F$

$ToHome^{\uparrow}: \neg ixAtHome \wedge \neg r \wedge (ixOpened \vee \neg gOnR) \;/\; qxToHome \coloneqq T; r \coloneqq T$
$ToHome^{\downarrow}: ixAtHome \;/\; \mathrm{r} \coloneqq F;\; qxToHome \coloneqq F$

b) För att styra med ladder oavsätt var vi befinner oss så behöver vi utgå ifrån vilka start-villkor varje operation behöver för att rätt sak skall ske. Tex som följande ladderkod: