Uppgift 2
The Smoker's Problem

Parallella processer G1F

Grupp A4

Adam Näslund a13adana
Victor Karlsson c12vicka
Alexander Milton b13alemi

Vi hann tyvärr inte skriva färdigt rapporten. Därför lämnar därför endast in en kopia av vad vi skrivit hittills och skulle uppskatta kritik på denna.

1 Inledning

Grunduppgiften bestod i att simulera en situation där ett antal individer står samlade runt ett bord. En av individerna är en agent och de övriga är rökare. Varje rökare har en oändlig mängd av en av de tre resurser som krävs för att röka, vilka är tobak, papper och tändstickor. Agenten har en oändlig mängd av samtliga resurser och lägger upp en enhet var av två olika slumpmässigt valda resurser på bordet. När de kompletterande resurser den behöver finns på bordet tar en rökare resurserna från bordet och börja röka. När en rökare tagit resurserna från bordet lägger agenten fram nya resurser.

I grunduppgiften finns en rökare av varje sort, det vill säga en som har tobak, en som har papper och en som har tändstickor. Två utökade versioner av grundproblemet gavs även, i den första finns ett godtyckligt antal rökare och i den andra finns dessutom krav på att en lösning garanterar en rättvis fördelning av resurserna (fairness) samt att alla rökare får röka någon gång (liveness).

Samtliga problem i uppgiften löstes med hjälp av programmeringsspråket SR och lösningen på det första problemet modellerades och verifierades även i verktyget UPPAAL.

2 Problembeskrivning

In this section, you shall describe the problem that you try to solve. It is the problem presented in the assignment but you must describe it in your own words and relate it to the classic problems. This is also the correct place to include a discussion about necessary properties of a solution (e.g., global invariant) and any kind of difficulties that you foresee.

Smoker's Problem behandlar en situation där ett antal rökare och en agent samarbetar för att producera cigaretter. Agenten förfogar över en oändlig mängd av de resurser som krävs för att producera en cigarett; tobak, papper och tändstickor. Agenten placerar iterativt två resurser av olika typer på bordet och meddelar med hjälp av meddelandepassering rökarna om vilka resurser som placerats på bordet. Rökarna har i sin tur en oändlig mängd av endast en resurstyp. Utan att kommunicera med varandra måste rökarna avgöra om de själva behöver de båda resurserna som ligger på bordet för att kunna producera en cigarett och i så fall plocka upp och konsumera dem.

2.1 Initialt program

I den första versionen av programmet ska enbart tre rökare befinna sig runt bordet, en rökare för varje typ av resurs. En och endast en rökare ska ta emot de två resurser som placeras på bordet. Det finns ingen problematik som rör konkurrens om företräde mellan rökarna då resurstypen skapar mutual exclusion mellan dem. Det finns heller inga krav på fairness, programmet behöver enbart simulera systematisk produktion och konsumtion.

Uppgiften kan liknas vid det klassiska problemet *Producer/Consumer* där två olika generella typer av processer existerar; en typ som producerar några eller någon typ av resurs och en annan typ som konsumerar de eller den resursen. Antalet processer oavsett typ kan variera (t.ex. kan flera producenter generera resurser till en ensam konsument eller vice versa). I det här faller agerar agenten som en typ av producent som kontinuerligt levererar två olika typer av resurser medan rökarna, trots att de förser sig själva med den tredje resursen, är agerar konsumenter i sammanhanget. Med hjälp av producentens resurser möjliggörs konsumtion hos rökarna.

2.2 Arbiträrt antal rökare

Programmet ska i den andra versionen av programmet stödja ett arbiträrt antal rökare som alla är i behov av olika resurstyper, även om flera rökare behöver samma två resurser. Det kommer alltid i alla delar av uppgiften finnas minst tre rökare, en för varje typ av resurs, men deluppgiften kräver även att flera rökare ska kunna plocka från bordet, trots att de konkurrerar om samma resurser. Den mest uppenbara utmaningen med problemet är att två eller fler rökare kan försöka plocka samma två resurser från bordet, men endast en skall kunna göra det.

2.3 Fairness

I den tredje och sista programversionen måste fairness etableras. Först av allt behöver rättvisa i sammanhanget av situationen definieras. Vad inne rättvis distribution av resurser? Ska alla involverade parter motta samma mängd resurser eller ska de enbart ha samma möjlighet till det, även om resurserna väljs ut slumpmässigt? Var går balansgången mellan uppgiftskrav och fairness? Och hur skapar man rättvisa när omständigheterna är slumpmässiga?

Ett antal globala invarianter kan etableras för att beskriva vilka förhållanden som måste hålla under hela programmets exekvering:

$$\forall ! \, deadlock$$

3 Metod

3.1 Design

In this section, you shall discuss your approach to solve the problems identified in previous section. Include the design decisions in a stepwise manner. The difference between a design decision and an implementation has to do with the level of details. For example, processes, responsibilities and communication (who talks with whom) are all design decisions.

3.1.1 Initialt program

Uppgiften löstes genom att implementera två processtyper, en agent som instansieras en gång samt en rökare om instansieras tre gånger. Varje rökare har ett eget index som identifierar den samt anger vilken typ av resurs som hålls av rökaren. Fyra kanaler skapas, en för varje rökare och en för agenten. Agenten har tillgång till samtliga kanaler medan rökarna enbart kan kommunicera med agenten på dess kanal.

Agenten slumpar fram två olika resurser och skickar ut en signal i form av en array till alla rökare. Arrayen innehåller information om vilka resurser som ligger på bordet. Eftersom att programmet använder sig av asynkron meddelandeöverföring är mottagning (*receive*) blockerande för rökarna. De väntar alltså på att ta emot ett meddelande från agenten. När de mottagit meddelandet undersöker de om deras egna föremål kompletterar de två som ligger på bordet och går i sådant fall vidare i sitt flöde. Om de redan höll någon av de resurser som erbjöds återgår de till att invänta ett nytt meddelande.

Om två resurser som en rökare saknar har meddelats som tillgängliga kommer rökaren att skicka tillbaks två signaler till agenten på agentens kommunikationskanal. Varje signal innehåller det id (ett heltal 1-3) som representerar resurstyperna rökaren avser att ta. Denna process motsvarar handlingen att plocka föremålen från bordet samt att direkt efteråt konsumera den färdiga cigaretten.

Agenten väntar i sin tur på att ta emot ett meddelande från någon rökare om att resurserna som placerats på bordet är bortplockade. Agenten tar emot en signal i taget tills alla resurser som plockats fram anmälts som borttagna. Efter det börjar agenten om sin loop och slumpar fram två nya värden medan rökaren som ”konsumerat” resurserna återgår till att invänta ett nytt meddelande.

3.1.2 Arbiträrt antal rökare

I den andra deluppgiften har den ursprungliga lösningen utvecklats en aning för att kunna stödja ett arbiträrt antal rökare som kräver samma resurstyper. En enkel variabel har lagts till som inkrementeras varje gång två nya resurser läggs upp. Det gör att programmet kan upprätthålla versionshantering likt ett kölappssystem med hjälp av variabeln.

Istället för att endast skicka ut ett meddelande till alla rökare skickar agenten även med versionsnumret. Rökare som behöver resurserna returnerar versionsnumret till agenten i en först-till-kvarn-manér. Agenten tar emot versionsnumret och kontrollerar om det stämmer överens med den aktuella versionen och meddelar då rökaren som skickade meddelandet att den har tillåtelse att ta resurserna och versionsnumret inkrementeras omedelbart för att etablera att nya resurser ska plockas fram.

Om en rökare skickar ett meddelande till agenten med ett versionsnummer som gått ut vet agenten att resurserna redan är utdelade och skickar en signal om att rökarens efterfrågan nekas.

Lösningen ser till att endast en rökare tar emot de två resurser som plockats fram och att inte två rökare med samma krav försöker plocka samma två resurser. Dock är lösningen argumenterbart orättvis då den rökare som först meddelar agenten alltid kommer att få resurserna. Det innebär att även om samma resurser slumpas fram varje gång finns det en risk att en rökare svälts ut.

3.1.3 Fairness och liveness

I den tredje och sista deluppgiften måste handhavandet av resurserna ske under definierat rättvisa förhållanden och lösningen garantera liveness. Uppgiftsbeskrivningen specificerar att agenten slumpmässigt ska välja vilka resurser som ska läggas på bordet. Detta är ej förenligt med liveness då det kan innebära att rökare av en särskild typ svälts eftersom de resurser de behöver inte läggs på bordet. Slumpmässigheten innebär även att fairness inte kan garanteras eftersom vissa resurser kan komma att läggas upp flera gånger än andra. Det var därför nödvändigt att frångå den delen av uppgiftsbeskrivningen för att kunna garantera liveness och fairness i lösningen.

Valet av resurser sker inte längre slumpmässigt, utan väljs istället av en algoritm. Det finns totalt tre olika kombinationer som iterativt väljs ut så att samma resurskombination förekommer lika ofta. Detta går emot den första uppgiftsbeskrivningen, men garanterar att ingen *typ* av process kan svältas.

Det nya hindret är att agenten inte känner till hur många processer som existerar. Rökare kan inte konkurrera om tillgängliga resurser genom att vara först till kvarn utan att riskera utsvältning av andra processer. Detta löses genom ett kösystem där varje rökare som inväntar *receive* skickar ett meddelande med sitt egna id till agenten på en kanal som är unik för varje resurstyp. Samtliga meddelanden buffras i agenten ”inkorg” av meddelanden och behandlas ett i taget i FIFO-ordning varje gång två resurser ska distribueras till en rökare.

På så vis garanteras alla rökare *eventual entry* då alla element som i teorin kan orsaka oändliga loopar eller baserar sig på ren slump är borttagna. Dock återstår ett problem. Agenten vet fortfarande inte hur många processer som existerar eller av vilken typ de är. Den kan därför inte avgöra hur processtyperna är proportionerade mot varandra, vilket har en potentiellt enorm inverkan på lösningens fairness. Det skulle mycket väl kunna finnas exempelvis 1 rökare som håller tobak, 1 rökare som håller papper och 100 rökare som håller i tändstickor. Om lika många av varje resurstyp hela tiden delas ut kommer en rökare som har tändstickor endast få röka en gång var trehundrade iteration, medan rökarna som ensamt håller i tobak och papper kommer att få röka var en gång var tredje iteration. Problemet kvarstår i vår lösning då vi valt att inte lägga till global funktionalitet och variabler som undersöker och uppskattar proportionen mellan processtyperna.

3.2 Verification decisions

In this section, you shall discuss which properties that your solution must have to be correct. ou shall also discuss how you are going to verify each of these properties, e.g., which tests to run and which queries to give the model checker.

4 Results

4.1 Implementation

The source code need not be included in the report since you are supposed to submit the sr files to your drop box. However, this is the place where you can discuss the code you have written. Do you have any comments to your source code?

4.2 Verification

This section contains your arguments for your solution meeting, or not meeting, the requirements. Are you confident with your solution? With what inputs did you test the program? To what extent have you used uppaal for verification? What queries did you run and what was the result of this? List your queries with the results you got from the model checker and explain what these results mean. For example: A[] !deadlock, result true, means that you have absence of deadlock. Given the verification, under what circumstances can you be sure that your program is correct?

5 Discussion

This section contains a discussion of whether the problem was successfully solved. Have you identified any problems with your solution? Can your solution be improved and if so how? Was it anything that surprised you during the verification? Note: it's seldom a good approach to conclude that the program is perfect. I want to see that you can estimate your success.

Första lösningarna är naiva och har varken fairness eller liveness. I lösning två är det race mellan de rökare som är av samma typ.

Lösning tre är mycket finare och bättre på alla sätt, kortare och enklare kod, garanterad liveness och fairness (på bekostnad av att slumpmässigheten tagits bort).