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 in av vad vi skrivit hittills och skulle uppskatta kritik på det.

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

Smoker's Problem beskriver en situation där ett antal rökare och en agent samarbetar för att producera cigaretter som rökarna sedan röker. Agenten förfogar över en oändlig mängd av de tre resurser som krävs för att producera en cigarett vilka är tobak, papper och tändstickor. Rökarna har en oändlig mängd av en av de tre resurserna och behöver en enhet var av de andra två för att kunna röka.

Agenten placerar en enhet var av två olika resurser på ett bord, rökare som vill använda resurserna begär tillgång till dem av agenten som ger resurserna till en rökare som sedan röker.

2.1 Grundproblem

I grundproblemet finns tre rökare, en som har tobak, en som har papper och en som har tändstickor. Det finns ingen problematik som rör konkurrens om företräde mellan rökarna då resurstypen skapar mutual exclusion mellan dem. Det finns heller inga krav på rättvis fördelning (fairness) eller på att alla rökare får röka (liveness).

Global invariant för en lösning av detta problem försäkrar att agenten placerar ut två olika resurser.

$$\forall i,j \in \text{resources}(\text{onTable}(i) \land \text{onTable}(j) \rightarrow \text{typeOf}(i) \neq \text{typeOf}(j))$$

2.2 Utökat problem: Godtyckligt antal rökare

I den första utökade versionen av problemet finns det ett godtyckligt antal rökare. Det kan därmed finnas flera rökare av varje sort och agenten måste se till att resurserna enbart delas ut till en rökare. Det finns dock fortfarande inga krav på fairness eller liveness i lösningen.

2.3 Andra utökade problem: Fairness och liveness

I den andra utökade versionen av problemet ställs, förutom tidigare krav, även krav på att alla rökare garanterat får röka (liveness) och att lösningen måste vara rättvis (fairness). Rättvisa kan tolkas på olika sätt men den tolkning som valdes var att samtliga rökare får röka lika många gånger; det vill säga att om det finns r antal rökare och agenten lägger fram resurser n gånger får varje rökare röka minst $\lfloor r/n \rfloor$ gånger.

3 Metodik

3.1 Design

Uppgiften är en variation på det klassiska problemet Producer/Consumer där två olika typer av processer existerar, en typ som producerar resurser och en som konsumerar de producerade resurserna. I det här fallet finns en producent i form av agenten och tre olika typer av konsumenter i form av smokers.

3.1.1 Grundproblem

Grundproblemet löstes genom att varje rökare har ett 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. Rökarnas kanaler används av agenten för att meddela vilka resurser den lagt på bordet och agentens kanal används av rökarna för att begära resurser.

Agenten slumpar fram två olika resurser och skickar till varje rökare ut ett meddelande med tre heltal som representerar hur många av de respektive resurserna som är tillgängliga.

Rökarna tar emot detta meddelande från agenten, kontrollerar om de resurser som finns tillgängliga är de de behöver. Om de tillgängliga resurserna är de rökaren behöver skickar den två meddelanden till agenten, ett för varje resurs, om att de tagits bort och börjar sedan röka.

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 är borttagna. Efter det börjar agenten om sin loop och slumpar placerar två nya resurser på bordet.

3.1.2 Utökat problem: Godtyckligt 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 variabel har lagts till som ökas varje gång nya resurser läggs fram. Denna variabel används för att garantera att resurserna endast tas bort en gång.

När agenten skickar ut vilka resurser som är tillgängliga skickar den 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 Andra utökade problem: 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 Resultat

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?

De första två lösningarna är naiva och har svagheter. Den sista lösningen är enkel och fin.

4.2 Verifiering

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?

Vi skapade en UPPAAL-modell av den sista lösningen men kunde inte få den att verifiera något. UPPAAL bara kör för alltid när vi försöker.

5 Diskussion

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).