# Uppgift 2
# The Smoker’s Problem

Parallella processer G1F

Grupp A4

Adam Näslund a13adana
Victor Karlsson c12vicka
Alexander Milton b13alemi

# 1 Inledning

Uppgiften bestod av 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 olika resurser på bordet. En rökare som saknar de två upplagda ingredienserna tar dem från bordet, rullar en cigarrette och börjat röka. När bordet är tomt lägger agenten fram två nya resurser.

I grunduppgiften fanns 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 där det fanns ett godtyckligt antal rökare och samt i den sista även krav på att lösningen garanterar en rättvis fördelning av resurserna (fairness) och att alla rökare får röka någon gång (liveness).

Samtliga problem i uppgiften löstes med hjälp av programmeringsspråket SR och asynkron meddelandeöverföring. Lösningen på det första problemet modellerades verktyget UPPAAL och dess korrekthet verifierades via modellen.

# 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 slumpmässigt en enhet var av två olika resurser på ett bord. En rökare som har den kompletterande resursen tar resurserna från bordet och röker. När bordet är tomt lägger agenten fram nya resurser på bordet.

## 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 inte placerar ut mer än två resurser samt att de utplacerade resurserna är av olika typ.

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

## 2.2 Utökat problem: Godtyckligt antal rökare

I den första utökade versionen av problemet finns ett godtyckligt antal rökare av varje sort. Det kan därmed finnas flera rökare av varje sort och lösningen måste garantera att resurserna då enbart delas ut till en rökare. Lösningen måste även hantera att det kan saknas rökare av en viss sort. Inte heller i detta problem finns några krav på fairness eller liveness i lösningen.

## 2.3 Andra utökade problem: Fairness och liveness

I den andra utökade versionen av problemet ställs, utöver tidigare krav, även krav på att alla rökare garanterat får röka (liveness) och att lösningen måste ge en rättvis fördelning (fairness). Rättvis fördelning kan tolkas på olika sätt men den tolkning som valdes var att det innebär 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 rökare.

### 3.1.1 Grundproblem

Grundproblemet löstes genom att varje rökare har ett index som identifierar dem 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 aktuell rökaren när dennes resurser är tillgängliga och agentens kanal används av rökarna för att bekräfta att de tagit emot agentens meddelande och börjar konsumera resurserna.

Agenten slumpar fram två olika resurser, beräknar vilken typ av rökare som behöver de resurserna och skickar en signal på motsvarande kanal. Rökaren som lyssnar på den kanalen tar emot meddelandet från agenten, meddelar agenten att den tagit resurserna och konsumerar dem sedan. När agenten tagit emot meddelandet tömmer den bordet och lägger fram nya resurser.

### 3.1.2 Utökat problem: Godtyckligt antal rökare

För att enkelt hantera ett godtyckligt antal rökare av varje sort lägger agenten inte längre fram resurser oberoende av rökarna. Istället begär rökarna resurser av agenten, som sedan behandlar deras begäranden, lägger fram de resurser de behöver och svarar till rökaren.

Agenten har en kanal för att ta emot begäranden om resurser och varje rökare har en egen kanal för att ta emot bekräftelser.

En rökare som vill ha resurser skickar ett meddelande till agenten med sitt id nummer och den resurs den redan har. Agenten tar emot meddelandet, tar fram de resurser rökaren behöver och svarar på rökarens egen kanal. När rökaren får ett meddelande från agenten om att dess resurser är tillgängliga konsumerar den resurserna. Agenten rensar sedan bordet och behandlar nästa begäran.

Denna lösning frångår till viss del den ursprungliga problemdefinitionen då resurserna inte längre väljs oberoende av rökarna eller slumpas fram, utan väljs ut baserat på rökarnas behov. I annat fall skulle en framslumpad kombination av resurser som inte efterfrågades av någon rökare orsaka en svår situation som skulle behöva hanteras särskilt för att inte resultera i ett dödläge (deadlock) där agenten väntar för alltid på en rökare som inte finns.

### 3.1.3 Andra utökade problem: Fairness och liveness

I det tredje och sista problemet tillkommer krav på att resurserna måste fördelas rättvist mellan rökarna (fairness) samt att alla rökare får röka (liveness).

Grundproblemet 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 slumpas fram. En rättvis fördelning av resurserna skulle även vara omöjlig att garantera om fördelningen var slumpmässig. Det var därför nödvändigt att frångå den delen av uppgiftsbeskrivningen för att kunna garantera liveness och fairness i lösningen.

I och med att resurserna i föregående lösning väljs ut och distribueras helt beroende på vilka rökare som begär dem och att alla rökare köar tillsammans i en gemensam FIFO-kö i agentens meddelandebuffer ger den lösningen inte bara en rättvis fördelning mellan individuella rökare, utan

garanterar även att varje rökare får röka (så länge som agentens meddelandebuffer är minst lika stor som antalet rökare). Därmed uppfyller den även kraven för det tredje problemet och ges som lösning även på det.

## 3.2 Verifiering

Den första lösningen hade som krav att det fanns exklusiv tillgång till resurserna på bordet mellan processerna. Efter att de placerats ut kan de endast plockas upp en gång utav en rökare (mutual exclusion). Detta garanteras implicit om enbart agenten har tillgång till bordet. Agenten får inte heller lägga upp flera resurser av samma sort eller fler än två resurser, vilket är lösningens globala invariant. Vidare bör lösningen aldrig kunna hamna i deadlock. Att lösningen ej kan hamna i deadlock, att resurserna läggs fram på ett korrekt sätt samt vissa andra egenskaper verifierades genom att lösningen modellerades i UPPAAL och frågor ställdes mot modellen. Testkörningar gjordes även för att kontrollera implementationens korrekthet.

De andra två lösningarna skiljer sig främst genom att de har stöd för ett godtyckligt antal rökar-processer, fairness och liveness, vilket alla är saker som är svåra att modellera eller verifiera med modeller. Därför modellerades de lösningarna inte utan testades enbart.

# 4 Resultat

## 4.1 Implementation

För att leva upp till sina krav frångår andra och tredje lösningarna beskrivningen på två sätt. Agenten slumpar inte vilka resurser som läggs på bordet och agenten lägger inte upp resurserna oberoende av rökarna, detta skulle krävt komplex kod för att hantera situationer då ingen rökare av en särskild typ finns samt skulle varit omöjligt att förena med den tredje lösningens krav på fairness och liveness.

Även om den inte specifikt utformades för detta uppfyller den andra lösningen både liveness och fairness då alla rökare köar i en FIFO-kö i agentens meddelandebuffer. Därför fanns ingen anledning att skriva någon särskild kod för den tredje lösningen utan koden för den andra lösningen användes.

Samtliga lösningar är enkla, med få synkroniseringspunkten och kort kod vilket minskar risken för subtila fel i implementationen.

## 4.2 Verifiering

Den första lösningens korrekthet verifierades genom att den modellerades i UPPAAL och att följande frågor ställdes mot modellen med positivt resultat.

- A.Initial --> A.BroadcastInfo

Verifierar liveness i agenten, efter att den startats kommer den kontakta en rökare.

- E<> (S1.Smoke || S2.Smoke || S3.Smoke)

Verifierar att någon rökare kommer få röka.

- A[] (A.items_on_table[TOBACCO] <= 1 && A.items_on_table[PAPER] <= 1 && A.items_on_table[MATCHES] <= 1 )

Verifierar att agenten aldrig kan lägga upp mer än en resurs av varje typ på bordet. Detta är en del av den globala invarianten.

- A[] (A.items_on_table[TOBACCO] + A.items_on_table[PAPER] + A.items_on_table[MATCHES]) <= 2

Verifierar att agenten aldrig lägger fram mer än två resurser på bordet. Detta är den andra delen av den globala invarianten.

- A[] !((A.smoker_type > 3) && (A.smoker_type < 1))

Verifierar att agenten alltid kommer fram till en giltig typ på den rökare den tänker skicka till.

- A[] !deadlock

Verifierar att lösningen aldrig kan hamna i deadlock.

De krav som fanns den första lösningen var mutual exclusion vid tillgång till bordet samt att lösningen ej skulle kunna hamna i deadlock. Då endast agenten har tillgång till bordet och det endast finns en agent garanteras mutual exclusion vid tillgång till bordet. Att lösningen inte kan hamna i deadlock verifierades via en förfrågning gentemot UPPAAL-modellen.

Då de sista lösningarna inte verifierades utan enbart testades kan deras korrekthet inte garanteras. Lösningarnas enkelhet och förståelighet tillsammans med de lyckade testresultaten gör det dock mycket troligt att de är korrekta.

# 5 Diskussion

Lösningen till första deluppgiften kan inte garantera liveness då den väljer resurser slumpmässigt vilket kan resultera i att processer svälts om de resurser de behöver ej slumpas fram. Lösningen hanterar även bara en rökare av varje sort; fler av samma typ leder till att rökare kan svältas och om det saknas rökare av någon typ riskerar lösningen hamna i deadlock. Den löser dock det angivna problemet, som inte tar hänsyn till dessa problem.

Våra queries verifierar med hjälp av UPPAAL Model Checker att lösningen till den första deluppgiften exekverar utan risk för deadlock och garanterad mutual exclusion, så länge exakt tre rökare existerar i programmet.

Även om den andra lösningen inte utformades med målet att ha liveness eller fairness resulterade designen i att den hade både dessa egenskaper. Därmed kunde den användas som lösning även på det tredje problemet. Till skillnad från den första lösningen väntar agenten på att rökarna begär resurser innan den skapar dem, vilket på ett enkelt och elegant sätt ger både fairness och liveness via agentens meddelandebuffer. Lösningen frångår dock problembeskrivningen något, då den anger att resurserna skall väljas ut slumpmässigt, något som omöjliggör rättvis distribuering av resurser och således även liveness och fairness.

Den andra och tredje lösningen har ej verifierats med formell modellering, utan enbart testats genom upprepad exekvering, där operationer, kommunikation och synkronisering i samtliga tester fungerat felfritt. Eftersom att felfri exekvering inte implicit påvisar felfri kod kan det inte påstås att de sista lösningarna bevisligen garanterar liveness, fairness eller safety.