**Labb 4**

Labb 4.1

Första fallet:

Task som ska utföras är make-delivery(one-ring).

Givet tillstånd:

undelivered(*one-ring*),

destination(*one-ring*, *mount-doom*),

delivery-speed(*one-ring*, *slow*),

at(*hobbit*, *bag-end*),

holding(*hobbit*, *one-ring*)

Det enda som vi kan göra i det givna tillståndet är att använda metoden

make-delivery(pkg, from, to, carrier). Om vi tänker oss en trädstruktur så kommer alltså

walk(carrier, from, to) och deliver(pkg, carrier, to) att vara barn till

make-delivery(pkg, from, to, carrier). Efter att ha utfört den primära tasken walk så befinner vi oss i tillstånd:

undelivered(*one-ring*),

destination(*one-ring*, *mount-doom*),

delivery-speed(*one-ring*, *slow*),

at(*hobbit*, *mount-doom*)

holding(*hobbit*, *one-ring*)

Sedan efter att vi har utfört den primära tasken deliver så befinner vi oss I tillstånd:

destination(*one-ring*, *mount-doom*),

delivery-speed(*one-ring*, *slow*),

at(*hobbit*, *mount-doom*)

Eftersom walk och deliver är primära tasks så är vi färdiga här, trädet består endast av 3 noder.

Andra fallet:

Fungerar liknande ovan med undantag att vi måste använda oss av den första make-delivery metoden som är listad på hemsidan, eftersom vi nu ska ha fast som delivery-speed.

Detta ger att enter-vehicle, travel, leave-and-deliver blir barn till make-delivery och att leave-vehicle samt deliver blir barn till leave-and-deliver. Vi har nu delat problemet till bara primära tasks. Detta ger att vi har 6 stycken noder totalt i trädet.

Summering av labb 4.2 och lab 4.3

Jag tyckte att det var enklare att bygga en HTN domän när man väl hade kommit in i rätt tänk. Det blev enklare på grund av att man kunde tänkte lite mer som funktioner i ett ”vanligt” programmeringsspråk, som Java, med rekursion och liknande.

Det som var svårt var just att förstå syntaxen med bland annat ”if else” grenarna i metoderna. Det löste sig dock genom att titta på de givna exemplen på hemsidan samt exempel från föreläsningen. Denna svårighet hade troligtvis blivit enklare om jag hade väntat med labben till efter föreläsningen av Jonas.

JHOP2 är väldigt snabb på att göra beräkningar jämfört med planerarna från första labbarna. Det kändes som att det inte fanns något som gjorde att JHOP2 blev långsam. Gjorde ett så stort problem så att det blev ”overflow error” tillslut, antar att det har med minnesbegränsning att göra.

Problemen som JHOP2 klarade av är avsevärt mycket större än de problem som gjordes i classical planning och klarar dessutom av att skapa en plan snabbt. Det bror väl dels på att vi tagit bort lådor och personer och istället bara säger hur många ”lådor” som finns i depån och hur många varje plats behöver. Detta för att JHOP2 stödjer aritmetik med siffror. Mycket bra grej!

Kör tiden skalar inte med storleken på problemet, det går nästan lika fort att köra ett litet problem som ett stort.