Labb X1: Jämförelse av Paradigm

Emilie Ruixin Cao & Nils Malmberg
ercao@kth.se & nilsmal@kth.se

12 mars 2025

1 Uppgifter

Här är uppgifterna på Kattis:

- Bus Numbers
- Calculating Dart Scores
- Peragrams

Uppgift	**Paradigm 1 (språk)**	**Paradigm 2 (språk)**
Bus Numbers	Imperativ (Java)	Funktionell (Haskell)
Dart Scores	Imperativ (Java)	Logisk (Prolog)
Peragrams	Funktionell (Haskell)	Logisk (Prolog)

Tabell 1: Fördelning av uppgifter på paradigm och språk

2 Dokumentation

2.1 Busnumbers i Java & Haskell

Java valdes för Bus Numbers i den imperativa lösningen eftersom det är ett strukturerat och välbekant språk för hantering av listor och sortering genom explicita loopar och villkor. Det ger full kontroll över iteration och array-manipulation, vilket passar uppgiftens krav på att identifiera och gruppera sekvenser av bussnummer. Haskell valdes för den funktionella lösningen eftersom det är utmärkt för listbehandling, sortering och pattern matching. Funktionella språk som Haskell gör det enkelt att arbeta med rekursion och högre ordningens funktioner, vilket ger en kort och elegant lösning på problemet.

Uppgiften löstes i Java före Haskell, och lösningen kunde i stora drag översättas. Exempelvis kunde man använda ungefär liknande steg (funktioner) för att lösa uppgiften, ett sådant exempel är att börja med en funktion som konverterar användarinputen till en lista av Int innan man behandlar listan av Int.

Tillvägagångssättet skiljer sig lite då haskell behövde använda rekursion för att gruppera på varandra följande heltal medan Java kunde använda sig av en loop för att iterera genom listan istället. I java formatteras listan med Int direkt till en lista av String genom att behandla alla sublistor med mer än 2 element så att de formatteras X-Y. I haskell så sparas allting som en lista som innehåller listor med Int. Sedan behandlar man alla element med mer än 2 i längd så att den formatteras X-Y.

Kattis submission ID för Java: 15542374
Kattis submission ID för Java: 15556559

2.2 Dart Scores i Java & Prolog

Prolog valdes för att lösa Dart Score eftersom det är väl lämpat för problemsökning och backtracking, vilket underlättar att hitta en kombination av kast som summerar till målvärdet. Genom att definiera regler för single, double och triple träffar kan Prolog effektivt utforska möjliga lösningar. Java valdes eftersom det erbjuder tydlig kontroll över iterationer och villkor, vilket gör det enkelt att systematiskt generera och validera möjliga kastkombinationer. Dessutom möjliggör Javas effektiva hantering av datastrukturer en snabb och strukturerad lösning av problemet.

Lösningarna skiljer sig en bit och kunde inte helt direktöversättas då prolog och java skiljer sig markant. I java skrevs hela lösningen direkt i mainfunktionen, och lösningen görs genom att spara alla möjliga kastnamn i throwNames och alla möjliga numeriska poäng i throwValues. Sedan checkar man om användarinputens värde motsvarar ett värde som redan finns i throwValues. Om inte så itererar man throwValues två gånger och ser om det fortfarande motsvarar en möjlig kastkombination av två kast. Om det ändå inte funkar så gör den tre iterationer av throwValues. Om inget funkar så printar den 'impossible' i terminalen. I prolog är det ett liknande tillvägagångssätt fast här har vi definierat flera funktioner istället. Vi har definierat ett predikat som heter throw, den tar in tre inparametrar: kasttyp(single, double, triple), N, och Score. I predikatet definierar vi N som ett värde mellan 1-20, och sedan definierar vi score som 1*N, 2*N, eller 3*N beroende på om det är single, double, eller triple. Sedan definierar vi ett predikat som heter find_combination som gör samma som java-lösningen. Den evaluerar först användarinputen för ETT kast, sedan två kast, och sist tre kast.

Kattis submission ID för Java: 15558635
Kattis submission ID för Prolog: 15559703

2.3 Peragrams i Haskell & Prolog

Haskell och Prolog valdes för denna uppgift på grund av deras styrka i att hantera problem relaterade till mönsterigenkänning och rekursion. Haskell, med sin funktionella natur och effektiva hantering av listor, gör det enkelt att analysera

bokstavsfrekvenser och beräkna de nödvändiga borttagen för att uppnå ett Peragram. Prolog, å andra sidan, är särskilt bra på att definiera regler för problem som involverar relationer, såsom att identifiera en anagramstruktur och kontrollera om en permutation kan bilda ett palindrome. Båda språken erbjuder eleganta och effektiva lösningar för denna typ av problem.

Tillvägagångssättet för att lösa uppgiften i dessa två paradigm var väldigt liknande. Vi kom fram till att ett peragram måste innehålla 0 eller 1 bokstäver som är ett udda antal, och detta var nyckeln till att lösa uppgiften. Inputsträngen måste alltså innehålla absolut max en bokstav med udda frekvens, och därmed så måste också minsta antal bokstäver som tas bort vara lika mycket som antalet bokstäver med udda frekvens minus ett. Detta tillvägagångssätt var i princip direktöversatt mellan de två paradigmen.

Kattis submission ID för Prolog: 16863350
Kattis submission ID för Haskell: 16840686

2.4 Avslutande reflektion

Som avslutande reflektion så kan man konstatera att de tre paradigmen har väldigt olika styrkor som gör de lämpliga i olika situationer. Både prolog och haskell är starka språk när det kommer till mönstermatchning och rekursion vilket är väldigt användbart vid bl.a. listbehandling. Det som vi anser är den största fördelen med imperativa språk är att man har full kontroll över allting, vilket gör att det blir mycket mer intuitivt hur man ska lösa diverse uppgifter. Däremot kan man observera på kattis-inlämningarna att java är betydligt långsammare än haskell och prolog.