PROGD X1

Felix Hedenström <fhede@kth.se>
Jonathan Rinnarv <rinnarv@kth.se>

8 maj 2016

Alphabet Spam

- C: 1152647
- Haskell: 1152644

Detta skrevs i C och Haskell. De valdes på grund av att Prologs djupsökande inte var nödvändig för denna uppgift. Man behöver inte utforska massa olika fall utan man ska iterativt eller rekursivt gå igenom ett bestämt antal karaktärer, något som både C och Haskell är väldigt bra på.

Paradigmreflektion

Vi tycker C är enklare, då vi är mer bekvämare. Input och output i C är också väldigt enkelt, medan Haskells statelessness gör att det blir lite krångligt. Vi löste det också olika i språken. C löste det iterativt medan Haskell löste det rekursivt. Det går att översätta lösningen i Haskell till C, då det är ganska enkelt att programera rekursivt i C, men att skriva iterativt i Haskell är inte lika lika lätt om man inte räknar med listcomprehensions.

Flexible Spaces

- Prolog: 1152638
- Haskell: 1152630

Detta program skrevs i både Haskell och Prolog. Dessa språk valdes då uppgiften var ställd som att hitta permutationer. I verkligheten märkte vi att vi inte behövde söka igenom kombinationer, utan bara lösa det rekursivt genom att stega igenom varje element i listan utav väggar. Lösningarna var nog ungefär lika komlplexa i båda språken, men vi kände oss bekvämare i Haskell vilket gjorde att den gick snabbare att skriva.

Paradigmreflektion

Lösningarna var som sagt ungefär likadana i båda språken. Efter vi skrivit ett Haskellprogram som dubbelt svansrekursivt gick igenom alla kombinationer av väggar. Sedan översatte vi detta till Prolog, vilket var väldigt lätt då man kan använda [H|T] för att gå igenom listor svansrekursivt.

Felix Hedenström
Jonathan Rinnarv

Pebble Solitaire

- C: 1152626
- Prolog: 1152618

Dessa program skapades i C och Prolog. Prolog valdes då det är bra på att söka igenom massa olika alternativ, och C valdes för att vi var bekväma i det och vi sparade det till denna uppgift då den verkade vara svårast.

Paradigmreflektion

Det var förvåningsvärt lika lösningar i Prolog och C. Vi startade med en board, gick igenom varje position tills vi kom till en pebble. Då gick kollade vi om pebblen kunde röra sig vänster eller höger, sparade ett temporärt boardstate som beskrev hur det så ut efter move:et, sedan kallade vi på orginalmetoden/-funktionen för att ta reda på minsta möjliga antalet pebbles.

Detta blev lite svårare i Prolog då vi använde svansrekursion och inte kunde då tillbaka efter man gått förbi ett element. Detta löstes genom att köra två svansrekursiva metoder, en som gick vänster till höger, och en som gick höger till vänster. Dessa splittade vid varje sökning.

Labbreflektion

Ett tämligen självklart resultat vi kom fram till var att det är lättare att skriva i språk man är van vid. Eftersom vi var bekväma i C och bekvämare i Haskell än Prolog gick det bättre att skriva i de språken för att sedan försöka översätta till det svårare Prolog. Här en en utvärdering av språken:

C

C är det bästa språket för iterativa lösningar, då man lätt kan använda for-loopar. Det fungerar även bra för rekursiva lösningar. Det är skönt att kunna spara i states och det känns naturligt att tänka på variabler som platser där saker är sparade istället för hur det är i Haskell och Prolog.

Haskell

Haskell kan ge funktioner som resultat, något som vi inte direkt använde oss av, men något som är bra. Det är också väldigt lätt att skriva svansrekursiva metoder p.g.a. (a:s) syntaxen. Om Haskell kompilerar, fungerar antagligen koden som den ska, för att Haskell har sådan rigorös koll på hur man skriver allt att man måste försäkra sig om att man har skrivit helt rätt.

Prolog

Prolog är otroligt bra på att söka igenom många möjliga vägar. Väldigt bra om man vill leta igenom många permutationer. Det är också ganska lätt att första när man läser det, om man har lite kunskap om logik och predikat. Predikaten

i sig är också unika, då både C och Haskell använder sig av funktioner. Detta kräver att man tänker på ett väldigt annorlunda sätt än i C och Haskell.