LINKÖPINGS UNIVERSITET

TDP007: Konstruktion av datorspråk

Utvecklarblogg
Seminarieuppgift 4

Författare

Viktor Rösler, vikro653@student.liu.se
Jim Teräväinen, jimte145@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Vårterminen 2021
Version 1.0

18 februari 2021

1 Uppgift 1

1.1 Tolkning

Uppgiften var att komplettera den givna koden `constraint_networks.rb` så att klasserna för addition och multiplikation fungerade som de skulle. Utöver det skulle man också skriva testfall för dessa klasser och se till att de klarade testerna.

1.2 Tillvägagångsätt

Vi började med att läsa igenom koden och försöka förstå hur den fungerade. Vi satt rätt länge och ponerade innan vi faktiskt försökte köra koden första gången. Därefter gjorde vi många små test med enkla beräkningar för att vidare fastställa funktionen. Vi tyckte allting verkade fungera tills vi ville strukturera om så att vi även kunde använda subtraktion och division. Då märkte vi att ändringar i `out` (resultat) delen av klassen inte slog igenom till resten av värdena. Vi fixade det igenom att skriva till 2 extra if satser som hanterade när a respektive b var de som saknade värde.

1.3 Lärdomar

Det stora som vi tagit med oss från båda dessa uppgifterna är att läsa och tolka andra personers kod. Det blev mycket handkörning* och funderande samtidigt som vi testade så små delar av koden som möjligt tills vi förstod ungefär hur allting fungerade.

**Att exekvera koden för hand på papper*

2 Uppgift 2

2.1 Tolkning

Denna uppgift gick ut på att ta koden som vi blev givna i `constraint-parser.rb` och få den att fungera som planerat. Vi tolkade det som att vi skulle kunna skriva ett godtyckligt algebraiskt uttryck med addition, subtraktion, multiplikation och division och minst 1 okänd och programmet skulle automatiskt skapa ett Constraint Network baserat på det. När man sedan angav värden på alla okända förutom 1 skulle den sista okända räknas ut automatiskt.

2.2 Tillvägagångsätt

Igen så började vi undersöka koden och fundera på hur den fungerade. Lite tester och handkörning av koden senare så var vi rätt säkra på funktionen (eller avsaknaden av). När vi kollat igenom detta började vi med att skriva funktionen `get_connector` vars mål är att alltid hämta ut den connector i constraints som är resultatet av uträkningen. Med `get_connectors` metoden hade vi ett problem med att `conn_a` eller `conn_b` argumenten ibland var en `ArithmeticConstraint` istället för en `Connector`, så vi använder vår `get_connector` metod för att byta ut constraints mot connectors. I `replace_conn` metoden utökades till att kunna byta ut olika delar av `exp`, istället för att alltid byta ut `exp.out`. Förutom det så ändrade vi matchningen i `Statement` så att det blev `:expr=:expr`, utan detta kunde vi inte matcha när det var plus eller minus i högerledet (se rad 271).

2.3 Lärdomar

Vi började rätt tidigt med att skriva väldigt stela lösningar som inte alls fungerade för många olika indata men med tiden så har de blivit mer och mer lösa utan att tappa sin egentliga funktion. Särskilt nöjda är vi med utvecklingen av funktionen `get_connector` som från början letade efter subtraktionstecken men nu i slutet fungerar med något så simpelt som att hämta den connectorn med längst namn.