2015

Logiskt-bevis validering i Prolog

LOGIK FÖR DATALOGER

LUCAS LJUNGBERG

ANTON RONSJÖ

Uppgiften i denna laboration handlade till största del om att konstruera ett Prolog program som tog in en fil med ett bevis som indata och som sedan kontroller att beviset är korrekt. Programmet skrevs med SWI-Prolog kompilatorn. En del i uppgiften var även skulle konstruera två icke-triviala logiska bevis – ett korrekt och ett icke-korrekt där båda även ska använda sig and antagandeboxar.

Konstruktionen av programmet

Konstruktionen av programmet började med att säkerställa att indata kunde hanteras på ett korrekt sätt. Indata är en viktig del av programmet då det är genom fil-indata det logiska beviset ska hanteras. Sedan stegades utvecklingen av programmet igenom del för del. Först och främst lades en kontroll in för att säkerställa att målet och sista bevisraden är identiska. Detta för att bekräfta att det logiska beviset leder till den tänkta slutsatsen. Detta gjordes genom att plocka ut den sista raden i hela beviset och matcha det med det givna målet. Nästa steg blev att lägga in en check för om en premiss verkligen är en premiss. Detta gjordes på ett simpelt sätt genom att ta in den antagna premissen och kolla om den också finns i listan av premisser. Efter den delen var klar så var det dags att börja med huvud-loopen. Det är en svans-rekursiv loop som går igenom rad för rad uppifrån ner. Regler och bevis kollas genom mönstermatchning, vilket betyder att om ett element i raden matchar mönstret hos parametern så anser prolog det vara en matchning och kallar den funktionen. Hittar prolog ingen matchning, så misslyckas körningen. Som ett exempel då, om det tredje elementet på en rad innehar värdet ”premise” så kan prolog matcha det och kalla den funktionen med matchande argument. Då i det nämnda fallet så plockar funktionen in den relevanta informationen och skickar den vidare och kollas beroende på vilken slutsats som dragits och även vilken regel som användes för att dra slutsatsen. Nästa del var nog den största och ganska självklart den mest komplicerade delen. Nämligen att kolla att den regeln som hade lett till den givna slutsatsen faktiskt är korrekt. Detta gjordes på liknande sett som hur slutsatsen drogs. Regeln som användes i den nuvarande raden skickas till regel-checkning och går igenom mönster matchning även där. Om den givna regeln hittar en matchning i parametrarna så kallas den funktionen. Detta tillåter att varje regel kan hanteras separat och checkar kan göras ingående för respektive regel. I början implementerades några regler på detta vis, men senare behövdes boxhantering implementeras då vissa regler är beroende av antaganden.

Boxhantering

Boxhanteringen hanteras genom att än en gång mönstermatcha en rad, och utnyttja att första raden i boxen innehåller ”assumption” som tredje element. Om detta hittas så börjar en ny svansrekursiv loop som itererar igenom boxen tills en tom rad hittas, då kommer ett basfall matchas och de rekursiva anropen går tillbaka till början av grenen och fortsätta därifrån. På så sätt så kan det garanteras att alla rader i beviset nås och kan evalueras.

Beviskontroll

Beviskontrollen i helhet bygger på tanken att allting antas vara sant men att allting ska kollas. Blir någonting fel, då säger programmet till och den stämplar det logiska beviset som fel. Så i grund och botten så kollar den allting tills den hittar något fel och programmet kommer då säga till och avbryta programmet. Dessa fel kan sträcka sig från att en matchning inte kan hittas, vilket då betyder att en icke-existerande regel används, till att reglerna är använda på fel sätt. Men även andra fel som kan uppkomma.

Hjälpfunktioner

Förutom de grundläggande funktionerna som programmet krävde skapades även en del hjälpfunktioner som användes för att spara kod eller förkorta vissa processer. Ett par exempel på dessa är en funktion som hittar raden i beviset med det givna radnumret. Detta görs genom att stega

igenom beviset tills raden är hittad. Ett annat exempel är en funktion som tar in två radnummer och returnerar värdet/slutsatsen på den raden så att det kan användas i regel-checken. Ett sista exempel är en funktion som konkatenerar två listor till en lista. Detta för att underlätta funktionen som hittar raden med det givna radnumret.

Problem

Givetvis så uppstår en del problem i ett arbete, framförallt i ett arbete av detta storlek, och dessa ansågs vara bra att notera för framtida behov. När arbetet av uppgiften var till stor del klar så upptäcktes ett par problem relaterade till hur vi hanterade boxar. Detta problem upptäcktes då hanteringar för regler som *eller elimination* som behöver ha tillgång till boxar andra regler inte ska ha tillgång till. Detta problem ansågs som ganska stort och en lösning (som vi redan innan insåg skulle bli ett problem) blev att göra en hjälpfunktion som söker igenom hela beviset efter den efterfrågade raden. Detta både skapar och uppenbarade ett problem gällande variabel-omfånget. Problemet med variabel-omfånget visade problemet hos implementationen av den svans-rekursiva loopen. Den lösningen som implementerades tillåter inte på ett rimligt vis att hålla koll på ett omfång och det ansågs då att de checkar vi kunde göra för att förhindra icke-tillåtna referenser var mer lämpligt än att skriva om programmet från början. Detta i sin tur ledde till att 5 tester gav fel resultat (5 utav de icke-korrekta testerna). Det ansågs då mer rimligt att istället förstå varför felen uppstod.

Predikat

Namn	Parametrar	När predikatet är sant
verify	InputFileName – Sökväg till indata-filen.	Predikatet är sant när alla tester i programmet returnerar sant. Den returnerar falskt om något i beviset är fel eller om läsningen av indata blir fel.
valid_proof	Goal – Bevisets mål Proof – Listan med beviset	Predikatet returnerar sant om det angivna målet överensstämmer med slutsatsen i sista raden av beviset.
iterate	1. Basfall då den givna listan är tom. 2. Prems – Premisserna för beviset [Head | Tail] – Listan för beviset. Använder en svansrekursiv metod. Proof – En hel lista med beviset.	1. Den returnerar sant i samtliga fall. Då fallet bara är till för att hantera basfall. 2. Returnerar sant om ingen iteration av bevislistan returnerar falskt. Returnerar falskt om programmet hittar något fel i bevislistan (Hanteras i separata funktioner).
box_iterator	1. Basfall då den givna listan är tom. 2. [BoxHead | BoxTail] – Lista med bevis från inuti boxen. Använder en svansrekursiv. Proof – En lista med hela beviset.	1. Den returnerar sant i samtliga fall. Då fallet bara är till för att hantera basfall. 2. Returnerar sant om samtliga tester i iterationen returnerar sant. Om ett test

		returnerar falskt så returnerar även predikatet falskt.
append_list	1. [] – Kollar att den givna listan är tom. Basfall List – Den andra listan som skickas in i funktionen. List – En oinitierad variabel som kommer returneras för att predikatet ska bli sant. Kommer att erhålla informationen i det tidigare argumentet. 2. [Head | FirstTail] – Den första listan av de två som ska sättas ihop. Använder en svansrekursiv lösning där basfallet hanteras med de tidigare argumenten. Vi plockar även ut huvudet då det elementet kommer lägga sig i början av den oinitierade listan. List2 – Den andra listan av de två som ska sättas ihop. Den oinitierade listan kommer erhålla informationen i denna lista i basfallet. [Head |NewTail] – Den oinitierade listan. Head kommer erhålla värdet i första argumentet för att uppehålla likheten hos variablerna.	1. Basfall. Returnerar sant i samtliga fall (Om predikatet används som tänkt). Annars kan den returnera falskt om inte alla givna listor är identiska. 2. Returnerar den konkatenerade listan av de två första givna och därav sant. Kan returnera falskt om tre argument ges då den kan då matcha om listorna är lika.
find_nth	N – Det radnummer i beviset som vill hittas Proof – En lista med beviset Row – Raden som söks	Predikatet returnerar sant om raden kan hittas. Om Row är odefinierad så returneras även raden i fråga. Predikatet returnerar falskt om raden inte kan hittas eller om Row är initierad och den inte matchar den rad som innehåller N.
check_proof	Prems – Lista med premisser [Nr, X, Rule] – En rad som innehåller en radsiffra, ett värde och en regel Proof – En lista med hela beviset	Den kollar om X kan matchas vi någon av de fall som definierats. Om den kan matcha en så returnerar den sant om den efterföljande regeln är korrekt använd. Annars returnerar den false.
check_rule	Nr – Radnummer på den givna regeln. Action – Värdet som bevisats med den givna regeln Regeln kommer matchas med de argument som skickats. X – Den första radnummer referensen Y – Den andra radnummer referensen Proof – En lista med beviset	Returnerar sant om den givna regeln använts på ett korrekt sätt. Den matchar den givna regeln med någon utav de definierade i parametrarna. Varje regel har sina krav för att de ska returnera sant. Dessa krav är definierade enligt regler för naturlig deduktion.

		Returnerar sant om regeln är korrekt använd. Annars returnerar den falskt. Kan även returnera falskt om någon given information inte existerar eller om den givna regeln inte finns.

Resultat

De resultat vi fick på de givna test-filerna var att alla gick igenom förutom 5 utav de som skulle returnera falskt. Dom returnerade då alltså sant trotts att de skulle returnera falskt. Detta är listan på resultatet:

valid01.txt passed
valid02.txt passed
valid03.txt passed
valid04.txt passed
valid05.txt passed
valid06.txt passed
valid07.txt passed
valid08.txt passed
valid09.txt passed
valid10.txt passed
valid11.txt passed
valid12.txt passed
valid13.txt passed
valid14.txt passed
valid15.txt passed
valid16.txt passed
valid17.txt passed
valid18.txt passed
valid19.txt passed
valid20.txt passed
valid21.txt passed

invalid01.txt passed
invalid02.txt passed
invalid03.txt passed
invalid04.txt passed
invalid05.txt passed
invalid06.txt passed
invalid07.txt passed
invalid08.txt passed
invalid09.txt passed
invalid10.txt passed
invalid11.txt passed
invalid12.txt passed
invalid13.txt passed
invalid14.txt passed
invalid15.txt passed
invalid16.txt passed
invalid17.txt passed
invalid18.txt passed
invalid19.txt **failed**
invalid20.txt **failed**
invalid21.txt passed

invalid22.txt passed
invalid23.txt passed
invalid24.txt passed
invalid25.txt **failed**
invalid26.txt **failed**
invalid27.txt passed
invalid28.txt **failed**