DD1351 Logik för dataloger
Laboration 2: Beviskontroll med Prolog

D. Gurov, A. Lundblad

1 december 2022

1 Introduktion

I den här laborationen ska du konstruera och implementera en algoritm som kontrollerar huruvida ett bevis skrivet i naturlig deduktion (som beskrivet i kursboken) är korrekt eller inte. Indata till ditt program är en sekvent och ett bevis, och programmet ska svara *"yes"* om det givna beviset är korrekt och visar att den givna sekventen gäller och *"no"* annars. Som du känner till utgörs naturlig deduktion av en uppsättning regler som beskriver när och hur nya formler kan härledas. Ett språk väl lämpat för att kontrollera att sådana regler följs är Prolog, vilket vi kommer använda i denna laboration.

2 Angreppssätt

Hur ett korrekt bevis i naturlig deduktion ska se ut defineras i figur 1. Ett bevis kontrolleras alltså lämpligtvis rad för rad genom att verifiera att premisserna för respektive regel är uppfyllda. Vad man behöver ha tillgång till för att avgöra om en regel är korrekt applicerad är vilka formler och boxar som härletts ovanför i beviset. Väljer du att gå igenom beviset uppifrån och ner, måste du bokföra vilka formler och boxar som redan kontrollerats. Väljer man att gå nerifrån och upp kan man finna informationen i den del av beviset som man har kvar att gå igenom.

En stor del av arbetet går ut på att se till att boxar hanteras på ett korrekt sätt, dvs att man kan referera till hela boxar men inte till individuella formler inuti en stängd box. Detta kan lösas på många sätt. En lösning är att se till att en box öppnas vid och endast vid ett antagande och därefter, temporärt, bortse från boxen tills alla rader i boxen kontrollerats. En annan

Ett bevis (eller härledning) till en sekvent är en formelsekvens:

- där varje formel antingen är:
  - en *premiss* i sekventen, eller
  - ett tillfälligt *antagande*, eller
  - resultatet av någon *regel* applicerad på formler som:
    - kommer tidigare i sekvensen, och
    - inte förekommer i någon avslutad box.
- där alla antaganden är friade, och
- som slutar i sekventens slutsats

Figur 1: Definitionen av ett bevis i naturlig deduktion.

lösning är att göra det rekursivt, genom att utnyttja det faktum att boxar i sig måste uppfylla samma krav som ett komplett bevis.

Samtliga regler från figur 1.2 på sid. 27 i boken, samt regeln *copy* ska hanteras. Den fullständiga listan över regler hittar du i appendix A.

2.1 Exempel

Antag att vi vill kontrollera att beviset i figur 2 är ett korrekt bevis för följande sekvent: $\neg\neg(p \to \neg p) \vdash \neg p$. Först kontrollerar vi att målet, dvs högerledet av sekventen, (i detta fall $\neg p$) står på sista raden i beviset. Därefter kontrollerar vi rad för rad som beskrivet i figur 3. I figuren låter vi understreck ("_") stå för värden som vi inte bryr oss om, precis som i Prolog.

3 Uppgifter

Laborationen består av följande uppgifter:

1. Skriv ett icke-trivialt korrekt bevis och ett icke-trivialt felaktigt bevis, båda med boxar.

$$\neg\neg(p \to \neg p) \vdash \neg p$$

1	$\neg\neg(p \to \neg p)$	premise
2	$p \to \neg p$	$\neg\neg$e 1
3	p	assumption
4	$\neg p$	$\to$e 3, 2
5	$\bot$	$\neg$e 3, 4
6	$\neg p$	$\neg$i 3-5

Figur 2: En sekvent och exempel på bevis.

2. Torrkörr din tänkta algoritm med penna och papper och visa hur den fungerar för dessa två bevis.

3. Implementera beviskontrolleringen i Prolog för bevis i formatet som definieras i appendix A.

4. Kör ditt program på bevisen som du hittat på själv och på alla fördefinierade testfall (se tips nedan). Notera att *testsviten ej är uttömmande*, så att alla tester passerar betyder inte att er lösning är korrekt.

5. Sammanställ resultaten och tillvägagångssättet i en rapport. Rapporten lämnas in och fungerar som underlag vid redovisningen. Rapporten ska vara strukturerad, välskriven, och heltäckande. Förutom en generell beskrivning av den valda beviskontrollalgoritmen och speciellt boxhanteringen, bör rapporten innehålla en tabell som listar namnen på era predikat samt när varje predikat är sant respektive falskt. Inkludera även programkoden och exempelbevisen i ett appendix.

6. Under redovisningen ska du kunna argumentera att lösningen är korrekt, och vara beredd på att besvara frågor om er lösning och följande frågor: skulle ditt beviskontrolleringsprogram kunna användas för att *generera* bevis för sekventer? Vad gäller för begränsade fall? Testa gärna att exekvera ert program med premisser och mål, men lämna beviset som en variabel, och se vad som händer.

1	$\neg\neg(p \to \neg p)$	premise	Rad 1 är OK eftersom $\neg\neg(p \to \neg p)$ finns bland mängden av premisser.
	$\vdots$		

1	$\neg\neg(p \to \neg p)$	premise	Rad 2 är OK eftersom $1 \quad \neg\neg(p \to \neg p) \quad \_$ finns ovanför i beviset vilket är kriteriet för att få applicera $\neg\neg$e. Radnumret ska stämma överens med det som är angivet efter regeln men vi inte bryr oss om *hur* $\neg\neg(p \to \neg p)$ härleddes.
2	$p \to \neg p$	$\neg\neg$e 1	
	$\vdots$		

1	$\neg\neg(p \to \neg p)$	premise	Rad 3 är OK då en box öppnats.
2	$p \to \neg p$	$\neg\neg$e 1	
3	p	assumption	
	$\vdots$		

1	$\neg\neg(p \to \neg p)$	premise	Rad 4 är OK då både $2 \quad p \to \neg p \quad \_$ och $3 \quad p \quad \_$ finns i beviset ovan, vilket är kriteriet för att få applicera $\to$e.
2	$p \to \neg p$	$\neg\neg$e 1	
3	p	assumption	
4	$\neg p$	$\to$e 3, 2	
	$\vdots$		

1	$\neg\neg(p \to \neg p)$	premise	Rad 5 är OK då både $3 \quad p \quad \_$ och $4 \quad \neg p \quad \_$ finns i beviset ovan, vilket är kriteriet för att få applicera $\neg$e.
2	$p \to \neg p$	$\neg\neg$e 1	
3	p	assumption	
4	$\neg p$	$\to$e 3, 2	
5	$\bot$	$\neg$e 3, 4	
	$\vdots$		

1	$\neg\neg(p \to \neg p)$	premise	Rad 6 är OK då en box på formen $3 \quad p \quad \_$ $\_ \quad \_ \quad \_$ $\vdots \quad \vdots \quad \vdots$ $\_ \quad \_ \quad \_$ $5 \quad \bot \quad \_$ finns i beviset ovan, vilket är kriteriet för att få applicera $\neg$i.
2	$p \to \neg p$	$\neg\neg$e 1	
3	p	assumption	
4	$\neg p$	$\to$e 3, 2	
5	$\bot$	$\neg$e 3, 4	
6	$\neg p$	$\neg$i 3-5	

Figur 3: Exempel på beviskontroll. Notera att hela exemplet lika gärna kan läsas baklänges om man väljer att gå igenom raderna nerifrån och upp.

4 Indataformat

Varje indatafil består av tre prologtermer:

1. En lista av premisser (vänstra delen av en sekvent)
2. Ett bevismål (högra delen av en sekvent)
3. Ett bevis i naturlig deduktion

Exempelsekventen och det tillhörande beviset i figur 2 skulle till exempel se ut på följande sätt:

```
% En lista av premisser (vänstra delen av sekventen.)
[neg(neg(imp(p, neg(p))))].

% Målet (högra delen av sekventen).
neg(p).

% Beviset
[
  [1, neg(neg(imp(p,neg(p)))),          premise    ],
  [2, imp(p, neg(p)),                   negnegel(1)],
  [
    [3, p,                              assumption ],
    [4, neg(p),                         impel(3,2) ],
    [5, cont,                           negel(3,4) ]
  ],
  [6, neg(p),                           negint(3,5)]
].
```

Inläsningen av indata är mycket enkel då formatet består av Prolog-termer. För att skapa ett predikat, `verify`, som läser en lista av premisser, ett mål och ett bevis från `InputFileName` och skickar det vidare till ett `valid_proof` predikat skriver ni

```
verify(InputFileName) :- see(InputFileName),
                         read(Prems), read(Goal), read(Proof),
                         seen,
                         valid_proof(Prems, Goal, Proof).
```

Om detaljerna i formatet är oklara, se beskrivningen i appendix A.

5 Tips

- Följande kommando startar GNU Prolog, laddar `beviskoll.pl`, testar predikatet `verify('input.txt')` och avslutar Prolog:

  ```
  gprolog --query-goal "['beviskoll'], verify('input.txt')" --query-goal halt
  ```

  För SWI-Prolog, använd i stället detta kommando:

  ```
  swipl -g "['beviskoll'], verify('input.txt')" -g halt
  ```

- Börja med lösa enkla exempel som

  ```
  [p].
  p.
  [[1, p, premise]].
  ```

  När detta fungerar lägger du lämpligtvis till funktionalitet för övriga regler och slutligen hantering av boxar.

- På Canvas finner du en uppsättning testfall. I allmänhet måste er lösning passera alla tester (det kan finnas undantagsfall, t.ex. om ni kan argumentera för att en korrekt lösning inte nödvändigtvis ska godta ett visst testfall). Testsviten är inte heller uttömmande, så en lösning är inte nödvändigtvis korrekt även om alla tester passerar. För att köra alla tester kan du kopiera hela test-katalogen till din labb-katalog och använda skriptet `run_all_tests` eller Prologprogrammet `run_all_tests.pl`. Så här kan förloppet se ut:

  ```
  $ cd tests
  $ prolog
  GNU Prolog 1.3.0
  By Daniel Diaz
  Copyright (C) 1999-2007 Daniel Diaz
  | ?- ['run_all_tests.pl'].
  compiling run_all_tests.pl for byte code...
  tests/run_all_tests.pl compiled, ...

  yes
  | ?- run_all_tests('../DIN_PROLOG_FIL.pl').
  compiling DIN_PROLOG_FIL.pl for byte code...
  valid01.txt passed
  valid02.txt passed
  ```

```
valid03.txt passed
...
```

- Genom att ge kommandot `trace.` innan du kör ditt program, instruerar du Prolog att skriva ut vad som anropas, vad som lyckas, och vad som inte lyckas. Använd kommandot `notrace.` för att stänga av funktionen.

- Tänk på att i vissa Prolog-tolkar som gprolog så måste definitionen av ett predikat vara "sammanhängande" i filen. Om du "delar upp" predikat på följande sätt:

  ```
  predA(...) :- ...  % Börja definera predA
  predB(...) :- ...  % Definera predB
  predA(...) :- ...  % Fortsätt definera predA (FUNGERAR EJ)
  ```

  kommer du få ett felmeddelande i stil med

  ```
  warning: discontiguous predicate pred/1 - clause ignored
  ```

- Om något test inte ger det förväntade resultatet, försök ta bort irrelevanta delar av indata (dvs konstruera ett minimalt motexempel) innan du avlusar med `trace`. Att stega igenom en stor Prolog-körning leder sällan någon vart.

- Ett annat sätt att avlusa ditt program är att lägga till debuggutskrifter. För att till exempel se till att Prolog har lyckats uppfylla ett visst predikat kan du lägga till `write('predikatet uppfyllt!')` i slutet av det predikatet.

A Syntax av indatafilerna

Detta appendix beskriver formatet av indatafilerna i detalj.

A.1 Propositioner

För att förenkla inläsning och hantering av indata är propositionerna givna som Prolog-termer i prefixnotation. Några exempel är listade nedan.

Infixnotation	Motsvarande Prolog-term
$p \to q$	`imp(p, q)`
$(p \vee q) \wedge r$	`and(or(p, q), r)`
$\neg p \to \bot$	`imp(neg(p), cont)`
$((p \wedge q) \wedge r) \wedge s$	`and(and(and(p,q),r),s)`

Formellt kan syntaxen för propositionerna beskrivas i s.k. *BNF–notation* på följande sätt:

$$\phi ::= \texttt{a} \mid \texttt{b} \mid \ldots \mid \texttt{z} \mid \texttt{and(}\phi\texttt{,}\phi\texttt{)} \mid \texttt{or(}\phi\texttt{,}\phi\texttt{)} \mid \texttt{imp(}\phi\texttt{,}\phi\texttt{)} \mid \texttt{neg(}\phi\texttt{)} \mid \texttt{cont}$$

A.2 Regelappliceringar

Regelappliceringarna (den högra kolumnen i bevisen) är skrivna med konnektivet, följt av `int` eller `el` och eventuellt ett eller flera heltal som beskriver vilka rader regelappliceringen syftar på. Tänk på att radreferensernas ordning spelar roll! Radreferenserna följer samma ordning som premisserna i reglerna från figur 1.2 i boken. För till exempel `andint(5,3)` gäller alltså att vänstra konjunkten ska återfinnas på rad 5 och den högra på rad 3.

Prolog	Logisk	Prolog	Logisk
`premise`	premise	`impel(x,y)`	$\to$e x,y
`assumption`	assumption	`negint(x,y)`	$\neg$i x-y
`copy(x)`	copy x	`negel(x,y)`	$\neg$e x,y
`andint(x,y)`	$\wedge$i x, y	`contel(x)`	$\bot$e x
`andel1(x)`	$\wedge\text{e}_1$ x	`negnegint(x)`	$\neg\neg$i x
`andel2(x)`	$\wedge\text{e}_2$ x	`negnegel(x)`	$\neg\neg$e x
`orint1(x)`	$\vee\text{i}_1$ x	`mt(x,y)`	MT x,y
`orint2(x)`	$\vee\text{i}_2$ x	`pbc(x,y)`	PBC x-y
`orel(x,y,u,v,w)`	$\vee$e x,y–u,v–w	`lem`	LEM
`impint(x,y)`	$\to$i x–y		

B Exempelkörningar

```
$ cat valid.txt
[imp(p, q), p].

q.

[
[1, imp(p,q), premise],
[2, p, premise],
[3, q, impel(2,1)]
].

$ prolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?- [pc].
compiling /pc.pl for byte code...
/pc.pl compiled

yes
| ?- verify('valid.txt').

true ?

yes
| ?-
```

(a) En kontroll av ett korrekt bevis.

```
$ cat invalid.txt
[imp(p, q), p].

q.

[
[1, imp(p,q), premise],
[2, q, assumption]
].


$ prolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?- [pc].
compiling /pc.pl for byte code...
/pc.pl compiled

yes
| ?- verify('invalid.txt').

no
| ?-
```

(b) En kontroll av ett felaktigt bevis.