DD1351 Logik för dataloger
Laboration 3: Modellprovning för CTL

D. Gurov A. Lundblad K. Palmskog

23 oktober 2020

1 Introduktion

En *modellprovare* (engelska: *model checker*) är ett programverktyg som kontrollerar om en temporallogisk formel ϕ gäller i ett visst tillstånd s i en given modell $\mathcal{M}$, dvs kontrollerar om $\mathcal{M}, s \models \phi$ gäller. Här utvecklas och testas en modellprovare för en delmängd av reglerna i temporallogiken CTL.

Modellprovning handlar egentligen om att *avgöra* validiteten. Anledningen till att vi, i denna labb, kan göra detta genom bevissökning är att bevissystemet som presenteras är sunt och fullständigt och tillåter *ändligt* många beviskträd (för en given formel och given modell med ändligt antal tillstånd).

Syftet med labbuppgiften är att de studerande ska lära sig att:

- bygga enkla men nyttiga programverktyg;
- *modellera* systembeteende med övergångssystem;
- *specificera* systembeteendeegenskaper med CTL;
- *verifiera* egenskaperna med verktyget man har byggd;

Vi behandlar följande delmängd av den temporala logiken CTL:

$$\begin{array}{rcl} \phi & ::= & p \mid \neg p \mid \phi \wedge \phi \mid \phi \vee \phi \\ & \mid & \mathsf{AX}\ \phi \mid \mathsf{AG}\ \phi \mid \mathsf{AF}\ \phi \\ & \mid & \mathsf{EX}\ \phi \mid \mathsf{EG}\ \phi \mid \mathsf{EF}\ \phi \end{array}$$

CTL-formlernas *semantik* är definierad i Definition 3.15 i kursboken (sid 211), medan *modeller* (också kallade transitionssystem eller övergångssystem) är

definierade i Definition 3.4 (sid 178). Viktiga *specifikationsmönster* finns i avsnitt 3.4.3 (sid 215).

Laborationen utförs och redovisas individuellt eller i par. Om du är osäker vad detta innebär, vänligen kontakta kursansvarig.

2 Ett bevissystem för CTL

I figur 1 presenteras bevissystemet för CTL som ska användas i modellprovaren. Det bygger på tanken att G- och F-formler kan "vecklas ut" och utvärderas *rekursivt*. En finess är att vi behöver ta hand om *slingor* i modellen för att garantera att bevissökningen *terminerar*. Observera att om man stöter på en slinga när en G-formel utvärderas betyder det "success", men för en F-formel "failure"! Upptäckandet av slingor (för G- och F-formler) görs med hjälp av en lista U som registrerar vid vilka tillstånd formeln redan har utvärderats.

Bevissystemet använder syntaktiska sekventer av formen

$$\mathcal{M}, s \vdash_U \phi$$

som exakt motsvarar den semantiska sekventen $\mathcal{M}, s \models \phi$, fast utrustad med listan U (för slingdetektering). I premisserna till A-reglerna betecknar $s_1, \ldots, s_n$ *alla* efterföljare till tillståndet s i modellen $\mathcal{M}$. I premisserna till E-reglerna, däremot, betecknar s' *någon* efterföljare till s.

På föreläsning 12 visas några bevisexempel i systemet.

3 Modeller i Prolog

Det finns många sätt att representera modeller i Prolog. För att underlätta testningen använder vi oss av representationen beskriven nedan.

I modellen $\mathcal{M} = (S, \rightarrow, L)$ skulle tillståndsmängden S t.ex. kunna representeras av en lista av möjliga tillstånd. Ur modellprövningssynpunkt används inte denna mängd och utelämnas därför helt.

Transitionsrelationen $\rightarrow$ (benämnd `T` i programskelettet sist i detta labbpek) representeras av grannlistor, dvs, för varje tillstånd s anges en lista

$$p \ \frac{-}{\mathcal{M}, s \vdash_{[]} p} \ p \in L(s) \qquad \neg p \ \frac{-}{\mathcal{M}, s \vdash_{[]} \neg p} \ p \notin L(s)$$

$$\wedge \ \frac{\mathcal{M}, s \vdash_{[]} \phi \qquad \mathcal{M}, s \vdash_{[]} \psi}{\mathcal{M}, s \vdash_{[]} \phi \wedge \psi}$$

$$\vee_1 \ \frac{\mathcal{M}, s \vdash_{[]} \phi}{\mathcal{M}, s \vdash_{[]} \phi \vee \psi} \qquad \vee_2 \ \frac{\mathcal{M}, s \vdash_{[]} \psi}{\mathcal{M}, s \vdash_{[]} \phi \vee \psi}$$

$$\mathsf{AX} \ \frac{\mathcal{M}, s_1 \vdash_{[]} \phi \quad \cdots \quad \mathcal{M}, s_n \vdash_{[]} \phi}{\mathcal{M}, s \vdash_{[]} \mathsf{AX}\, \phi}$$

$$\mathsf{AG}_1 \ \frac{-}{\mathcal{M}, s \vdash_U \mathsf{AG}\, \phi} \ s \in U \qquad \mathsf{AF}_1 \ \frac{\mathcal{M}, s \vdash_{[]} \phi}{\mathcal{M}, s \vdash_U \mathsf{AF}\, \phi} \ s \notin U$$

$$\mathsf{AG}_2 \ \frac{\mathcal{M}, s \vdash_{[]} \phi \qquad \mathcal{M}, s_1 \vdash_{U,s} \mathsf{AG}\, \phi \ \cdots \ \mathcal{M}, s_n \vdash_{U,s} \mathsf{AG}\, \phi}{\mathcal{M}, s \vdash_U \mathsf{AG}\, \phi} \ s \notin U$$

$$\mathsf{AF}_2 \ \frac{\mathcal{M}, s_1 \vdash_{U,s} \mathsf{AF}\, \phi \quad \cdots \quad \mathcal{M}, s_n \vdash_{U,s} \mathsf{AF}\, \phi}{\mathcal{M}, s \vdash_U \mathsf{AF}\, \phi} \ s \notin U$$

$$\mathsf{EX} \ \frac{\mathcal{M}, s' \vdash_{[]} \phi}{\mathcal{M}, s \vdash_{[]} \mathsf{EX}\, \phi} \qquad \mathsf{EG}_1 \ \frac{-}{\mathcal{M}, s \vdash_U \mathsf{EG}\, \phi} \ s \in U$$

$$\mathsf{EG}_2 \ \frac{\mathcal{M}, s \vdash_{[]} \phi \qquad \mathcal{M}, s' \vdash_{U,s} \mathsf{EG}\, \phi}{\mathcal{M}, s \vdash_U \mathsf{EG}\, \phi} \ s \notin U$$

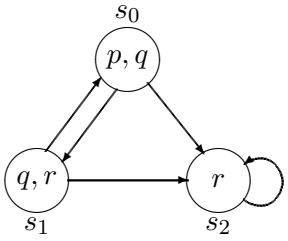
$$\mathsf{EF}_1 \ \frac{\mathcal{M}, s \vdash_{[]} \phi}{\mathcal{M}, s \vdash_U \mathsf{EF}\, \phi} \ s \notin U \qquad \mathsf{EF}_2 \ \frac{\mathcal{M}, s' \vdash_{U,s} \mathsf{EF}\, \phi}{\mathcal{M}, s \vdash_U \mathsf{EF}\, \phi} \ s \notin U$$

Figur 1: Ett bevissystem för CTL.

$s_1, \ldots, s_n$ som specificerar de efterföljande tillstånden till s.

Sanningstilldelningen L specificeras genom att man för varje tillstånd anger i en lista vilka variabler som håller i det tillståndet.

Exempel (motsvarar boken, sid 179, figur 3.3):



$\mathcal{M} = (S, \rightarrow, L)$ där

$S = \{s_0, s_1, s_2\}$

$\rightarrow = \{(s_0, s_1), (s_0, s_2), (s_1, s_0), (s_1, s_2), (s_2, s_2)\}$

$L = \{s_0 : \{p, q\}, s_1 : \{q, r\}, s_2 : \{r\}\}$

```
% States are s0, s1 and s2

% Adjacency lists of LTS
[
    [s0, [s1, s2]],
    [s1, [s0, s2]],
    [s2, [s2]]
].

% Labeling of LTS
[
    [s0, [p,q]],
    [s1, [q,r]],
    [s2, [r]]
].
```

Formler representeras på samma sätt som i labb 1: små bokstäver är variabler, och således även formler. Om `F` och `G` är formler så är `and(F,G)`, `or(F,G)`, `ax(F)`, `ag(F)`, `af(F)`, `ex(F)`, `eg(F)` och `ef(F)` också giltiga formler.

4 Uppgifter

Labbuppgiften består av följande deluppgifter:

1. **Verktygsutveckling**. Skriv en modellprovare för CTL i Prolog som gör bevissökning i bevissystemet från figur 1. Observera att om ni avviker från reglerna i bevissystemet så måste ni argumentera för att det nya systemet är sunt. Ett programskelett hittar ni sist i detta labbpek.

2. **Modellering**. Tänk på något system med datalogisk relevans som har ett icke-trivialt beteende. Konstruera en modell $\mathcal{M}$ till beteendet,

och rita dess tillståndsgraf. Helt abstrakta modeller som bara har tillståndsnamn som s_0 och s_1 är inte att betrakta som meningsfulla, och icke-triviala modeller har rimligen fem eller fler tillstånd. Ge en intuitiv förklaring till beteendet som modelleras och ditt val av atomer. Skapa en Prolog-kompatibel representation (som beskrivet ovan) av modellen.

3. **Specifiering**. Konstruera minst två icke-triviala och meningsfulla systemegenskaper uttryckta som CTL-formler relaterade till er modell, en som håller och en som inte håller. Förklara vilka beteendeegenskaper de formaliserar. Tänk på att man ska utgå bara från atommängden när man formaliserar egenskaper, och inte från något konkret modell.

4. **Verifiering**. Kontrollera med ert modellprovare att systemegenskaperna håller eller inte håller som förväntat. Kör dessutom alla fördefinierade testfall (se tips nedan).

5. **Rapport**. Sammanställ alla resultat i en rapport. Rapporten lämnas in och fungerar som underlag vid redovisningen. Rapporten ska vara strukturerad, välskriven, och heltäckande.

   Inkludera även en tabell i rapporten som listar namnen på era predikat och när vardera predikat är sant respektive falskt samt ett appendix med programkoden, exempelmodellen, och formlerna som formaliserar beteendeegenskaperna.

6. **Redovisning**. Förutom diskutera hur ni implementerat bevissystemet ska ni vara redo att svara på följande frågor:

   (a) Vad skiljer labbens version av CTL från bokens version? Hur kan man utöka modellprovaren så att den hanterar bokens CTL?

   (b) Hur hanterade ni variabelt antal premisser (som i AX-regeln)?

   (c) Hur stora modeller och formler går det att verifiera med ert modellprovare?

5 Tips

- På Canvas finns en uppsättning tester. Det är inte ett strikt krav att alla tester ska passera, men i de fall ett test inte passerar ska ni i detalj

kunna redogöra varför. För att köra alla tester kan ni kopiera hela testkatalogen till er labbkatalog 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...
valid001.txt passed
valid003.txt passed
valid007.txt passed
valid013.txt passed
...
```

- Genom att ge kommandot `trace.` innan ni kör ert program instruerar ni Prolog att skriva ut vad den gör, vad det lyckas bevisa och vad den misslyckas med att bevisa. Använd kommandot `notrace.` för att stänga av utskrifterna.

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

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

  kommer ni 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 ni debuggar med `trace`. Att stega igenom en stor Prolog-körning leder sällan någon vart.

- Ett sätt att avlusa ert program är att lägga till utskrifter. För att till exempel se att en regel EF_1 har applicerats kan ni lägga till `write('used EF1\n')` sist i implementationen av EF_1.
- Om ert program verkar hamna i en oändlig slinga, säkerställ att ni lagt in kod för att kontrollera regelvillkoren $s \in U$ och $s \notin U$ överallt där det behövs.

6 Programskelett

```
% For SICStus, uncomment line below: (needed for member/2)
%:- use_module(library(lists)).

% Load model, initial state and formula from file.
verify(Input) :-
    see(Input), read(T), read(L), read(S), read(F), seen,
    check(T, L, S, [], F).

% check(T, L, S, U, F)
%     T - The transitions in form of adjacency lists
%     L - The labeling
%     S - Current state
%     U - Currently recorded states
%     F - CTL Formula to check.
%
% Should evaluate to true iff the sequent below is valid.
%
% (T,L), S  |-    F
%             U

% To execute: consult('your_file.pl'). verify('input.txt').

% Literals
%check(_, L, S, [], X)      :- ...
%check(_, L, S, [], neg(X)) :- ...

% And
%check(T, L, S, [], and(F,G)) :- ...

% Or
% AX
% EX
% AG
% EG
% EF
% AF
```