Laboration 2: Modellprovning för CTL

Stefan Åhman
900326-2376
sahman@kth.se

Marcus Wallstersson
880301-6099
mwallst@kth.se

December 4, 2011

KTH Kista, Stockholm

Innehållsförteckning

1 Inledning

För att kunna kontrollera om en temporallogisk formel ϕ gäller i ett visst tillstånd s i en given modell $\mathcal{M}$ kan man använda sig av en modellprovare. Detta programverktyg måste i denna laboration implementeras att hantera följande delmängd CTL-reglerna (Computation tree logic):

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

$$\phi ::= p \mid \neg p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \mathsf{AX}\ \phi \mid \mathsf{AG}\ \phi \mid \mathsf{EX}\ \phi \mid \mathsf{EG}\ \phi \mid \mathsf{EF}\ \phi$$

Modellen som ska kontrolleras kan beskrivas med en tillståndsgraf, där CTL används för att sätta upp villkor som måste uppfyllas av tillståndsgrafen samt tillstånden. Uppkomsten av oönskade stigar kan undvikas med specifika regler. Detta kan göras i denna laboration med bevissökning då bevissystemet som används är sunt och fullständigt och tillåter ändligt många bevisträd.

2 Problem och Syfte

Syftet med laborationsuppgiften är att:

- Fördjupa försåelsen för CTL och hur temporallogik kan användas för att specifiera viktiga systemegenskaper [HR04].
- Lära sig använda Prologs sökteknik för bevissökning.
- Lära sig bygga enkla men nyttiga programverktyg som kan användas till systemverifikation.

3 Genomförande

Modellprovaren skrevs i prolog då det är ett lämpligt programmeringsspråk för bevissökning. De befintliga reglerna för CTL implementerades. Vissa av reglerna kräver variabelt antal premisser och detta måste hanteras av programmet. Implementationen av reglerna och modellprovaren gås igenom i kapitel 3.1. I kapitel 3.2 beskrivs vår egenvalda modell som föreställer ett trafikljus.

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

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

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

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

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

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

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

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

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

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

Figure 1: Regler för CTL

3.1 Modellprovaren

För att kunna testa modellprovaren fanns flertalet tester att tillgå som bestod av en liststruktur för att beskriva tillståndens egenskaper och grannar, detta beskrivs tydligare under Modell. Programmet skrevs så att en funktion “check” anropades med följande inparametrar:

```
check(T, L, S, U, F)
T - Alla tillstånd och dess grannar i listform
L - Lista över egenskaper i varje tillstånd
S - Aktuellt tillstånd
U - Lista för besökta tillstånd
F - CTL formel som ska testas
```

Check skrevs så att den med pattern matching kan matchas mot alla de regler som skulle implementeras. De matchades på följande sätt: `X`, `neg(X)`, `and(F,G)`, `or(F,G)`, `ax(X)`, `ag(X)`, `ex(X)`, `eg(X)`, `ef(X)`.

Nedan följer ett utrag ur programkoden för kontroll av `ef(X)`:

```
1  % EF 1
2  check(T, L, S, U, ef(X)) :-
3    not(member(S, U)),
4    check(T, L, S, [], X).
5
6  % EF 2
7  check(T, L, S, U, ef(X)) :-
8    not(member(S, U)),
9    member([S, Srest], T),
10   echeck(T, L, Srest, [S|U], ef(X)).
```

Då check stötte på `ef(X)` försökte den först med implementationen EF1 och sedan om den evaluerades till false försökte den med EF2.

EF1 skrevs så att den alltid kontrollerar att nuvarande tillstånd S inte finns bland tidigare besökta U och fortsätter sedan rekursivt med resten av beviset X och tömd lista U för tidigare besökta tillstånd. Detta uppfyller kraven för regel EF1 som kan ses i figur 1.

EF2 skrevs så att den på samma sätt som EF1 kontrollerar att S inte tidigare har besökts. I nästa steg kontrollerar den vilka grannar S har övergångar till och skickar med dessa till funktionen echeck. Denna funktion kontrollerar att någon av tillståndets S grannar evalueras till sant. Till echeck skickas även en lista innehållandes tidigare besökta tillstånd där nuvarande tillståndet S läggs till. Detta uppfyller kraven för EF2.

De resterande reglerna från figur 1 implementerades på liknande sätt och dessa kan ses i den bifogade koden under kapitel 5.

3.2 Modell

4 Resultat

5 Bilagor

```
1  :- use_module(library(lists)).
2  % Load model, initial state and formula from file.
3  verify(Input) :-
4      see(Input), read(T), read(L), read(S), read(F), seen,
5      check(T, L, S, [], F).
6
7  % check(T, L, S, U, F)
8  %      T - The transitions in form of adjacency lists
9  %      L - The labeling
10 %      S - Current state
11 %      U - Currently recorded states
12 %      F - CTL Formula to check.
13
14 % p
15 check(_, L, S, [], X) :-
16   member([S,Srest], L),
17   member(X,Srest).
18 % neg p
19 check(_, L, S, [], neg(X)) :-
20   member([S,Srest], L),
21   not(member(X,Srest)).
22
23 % And
24 check(T, L, S, [], and(F,G)) :-
25   check(T, L, S, [], F),
26   check(T, L, S, [], G).
27
28 % Or 1
29 check(T, L, S, [], or(F,_)) :-
30   check(T, L, S, [], F).
31 % Or 2
32 check(T, L, S, [], or(_,G)) :-
33   check(T, L, S, [], G).
34
35 % AX
36 check(T, L, S, [], ax(X)) :-
37   member([S,Srest],T),
38   acheck(T, L, Srest, [], X).
39
40 % EX
41 check(T, L, S, [], ex(X)) :-
42   member([S,Srest],T),
43   echeck(T, L, Srest, [], X).
44
45 % AG 1
46 check(_, _, S, U, ag(_)) :-
47   member(S,U).
48
49 % AG 2
50 check(T, L, S, U, ag(X)) :-
51   not(member(S,U)),
52   member([S,Srest],T),
53   check(T, L, S, [], X),
54   acheck(T, L, Srest, [S|U], ag(X)).
55
56 % EG 1
57 check(_, _, S, U, eg(_)) :-
58   member(S,U).
```

```
59
60 % EG 2
61 check(T, L, S, U, eg(X)) :-
62   not(member(S,U)),
63   member([S,Srest],T),
64   check(T, L, S, [], X),
65   echeck(T, L, Srest, [S|U], eg(X)).
66
67 % EF 1
68 check(T, L, S, U, ef(X)) :-
69   not(member(S,U)),
70   check(T, L, S, [], X).
71
72 % EF 2
73 check(T, L, S, U, ef(X)) :-
74   not(member(S,U)),
75   member([S,Srest],T),
76   echeck(T, L, Srest, [S|U], ef(X)).
77
78 % AF 1
79 check(T, L, S, U, af(X)) :-
80   not(member(S,U)),
81   check(T, L, S, [], X).
82
83 % AF 2
84 check(T, L, S, U, af(X)) :-
85   not(member(S,U)),
86   member([S,Srest],T),
87   acheck(T, L, Srest, [S|U], af(X)).
88
89 %%% Helper functions %%%
90
91 acheck(_, _, [], _, _).
92 acheck(T, L, [S|Sl], U, X) :-
93   check(T, L, S, U, X),
94   acheck(T, L, Sl, U, X).
95
96 echeck(T, L, [S|_], U, X) :-
97   check(T, L, S, U, X).
98 echeck(T, L, [_|Sl], U, X) :-
99   echeck(T, L, Sl, U, X).
100
101 not(P) :- call(P), !, fail.
102 not(_).
```

lab2.pl

Referenser

[HR04] Michael Huth och Mark Ryan (2004). *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge University Press, second utgåvan.