DD1351 LAB3 2018
DIEGO LEON 961206
VIKTOR MEYER 971203

Rapport Modellprovning CTL

Övergripande implementation

Enligt labblydelsen används ett indata format som består av en lista med närliggande LTS, en lista med märkning av LTS, samt en formel att pröva. Prologprogrammet som vi ska bygga och uppgiften för den här labben är alltså att ta fram ett modellprövningssytem för CTL.

Modellprövning handlar överlag om att avgöra validiteten hos en modell. En modell kan sägas vara valid om en given modell i sig är validerbar genom de angivna reglerna för CTL, reglerna som kommer att användas relaterar direkt till kursboken.

De mer tekniska delarna av implementationen sker i Prolog med hjälp av predikatlogik. Modellprövningsprogrammet startar som tidigare nämnt med indata i form av en lista med närliggande LTS, en lista med märkning av LTS samt en formel att pröva. Med hjälp av dessa indata har programmet i uppgift att pröva om formeln är faktiskt är validerbar eller ej.

```
Indata:
LTS Grannlista - Lista av alla närliggande grannar
LTS Märkning - Mål för beviset

Utdata:
Yes / No, där Yes motsvarar att det går att utföra beviset och Nej
motsvarar inte går att utföra beviset
```

Härledning av predikat

Literals

```
check(T, L, S, [], X) :- member([S, K], L), member(X, K).
```

Predikatet är sant om och endast om märkningen av den aktuella noden S innehåller formeln X.

Negation

```
check(T, L, S, [], neg(X)) :- \+ check(T, L, S, [], X).
```

Predikatet är sant om och endast om märkningen av den aktuella noden S inte innehåller formeln X..

And

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

Predikatet är sant om och endast om predikaten för formlerna F och G båda är sanna samtidigt.

DD1351 LAB3 2018
DIEGO LEON 961206
VIKTOR MEYER 971203

Or 1

```
check(T, L, S, [], or(F,G)) :- check(T, L, S, [], F).
```

Predikatet är sant om och endast om predikatet för F är sant.

Or 2

```
check(T, L, S, [], or(F,G)) :- check(T, L, S, [], G).
```

Predikatet är sant om och endast om predikatet för G är sant.

AX

```
check(T, L, S, [], ax(X)) :- member([S, Adj], T), checkAX(T, L, Adj, X).
```

Predikatet är sant om och endast om predikatet för formeln X är sant i alla närliggande noder till S.

EX

```
check(T, L, S, [], ex(X)) :- member([S, Adj], T), member(Nstate, Adj), check(T, L,
Nstate, [], X).
```

Predikatet är sant om och endast om predikatet för formeln X är sant i någon närliggande nod till S.

AG 1

```
check(T, L, S, U, ag(X)) :- member(S, U).
```

Predikatet är sant om och endast om det aktuella tillståndet redan är validerat, det vill säga S tillhör listan U vilket innebär att en slinga har upptäckts.

AG 2

```
check(T, L, S, U, ag(X)) :- \+ member(S, U), check(T, L, S, [],
X), member([S, Adj], T), iterAdj(T, L, Adj, [S|U], ag(X)).
```

Predikatet är sant om och endast om predikatet för formeln X är sant i alla tillstånd för alla vägar från S.

EG 1

```
check(T, L, S, U, eg(X)) :- member(S, U).
```

Predikatet är sant om och endast om det aktuella tillståndet redan är validerat, det vill säga S tillhör listan U vilket innebär att en slinga har upptäckts.

EG 2

```
check(T, L, S, U, eg(X)) :- \+ member(S, U), check(T, L, S, [], X), member([S,
Adj], T), member(Node, Adj), check(T, L, Node, [S|U], eg(X)).
```

Predikatet är sant om och endast om predikatet för formeln X är sant i alla tillstånd för någon väg från S.

DD1351 LAB3 2018
DIEGO LEON 961206
VIKTOR MEYER 971203

EF 1

```
check(T, L, S, U, ef(X)) :- \+ member(S, U), check(T, L, S, [], X).
```

Predikatet är sant om och endast om det aktuella tillståndet inte är validerat, det vill säga S tillhör inte listan U vilket innebär att ingen slinga har upptäckts.

EF 2

```
check(T, L, S, U, ef(X)) :- \+ member(S, U), member([S, Adj], T), member(Node,
Adj), check(T, L, Node, [S|U], ef(X)).
```

Predikatet är sant om och endast om predikatet för formeln X är sant något tillstånd för någon väg från S.

AF 1

```
check(T, L, S, U, af(X)) :- \+ member(S, U), check(T, L, S, [], X).
```

Predikatet är sant om och endast om det aktuella tillståndet inte är validerat, det vill säga S tillhör inte listan U vilket innebär att ingen slinga har upptäckts.

AF 2

```
check(T, L, S, U, af(X)) :- \+ member(S, U), member([S, Adj], T), iterAdj(T, L,
Adj, [S|U], af(X)).
```

Predikatet är sant om och endast om predikatet för formeln X är sant något tillstånd för alla vägar från S.

Modellering

```
[[s0, [s0, s1]],
 [s1, [s2]],
 [s2, [s3]],
 [s3, [s1, s4]], [s4, [s4]]].

[[s0, [w]],
 [s1, [q]],
 [s2, [q,p]],
 [s3, [q,p,c]], [s4, [r]]].

s0.

ef(r).
```

Ovanstående är en tänkt modell med atomerna (w=wait, q = player 0, p = player 1, c = coordinator, r = result). Modellen är tänkt att modellera ett spel mellan två spelare, modellen börjar i ett waiting tillstånd, fortsätter till en spelsession mellan de två spelarna samt avslutar med ett resultat.

DD1351 LAB3 2018
DIEGO LEON 961206
VIKTOR MEYER 971203

Sann formel

```
[[s0, [s0, s1]],
 [s1, [s2]],
 [s2, [s3]],
 [s3, [s1, s4]], [s4, [s4]]].

[[s0, [w]],
 [s1, [q]],
 [s2, [q,p]],
 [s3, [q,p,c]], [s4, [r]]].

s0.

ef(r).
```

Frågan ‘Kan spelet avslutas med ett resultat’ formulerar vi med hjälp av den logiska formeln ‘ef(r)’, vilket i detta fall är sant.

Falsk formel

```
[[s0, [s0, s1]],
 [s1, [s2]],
 [s2, [s3]],
 [s3, [s1, s4]], [s4, [s4]]].

[[s0, [w]],
 [s1, [q]],
 [s2, [q,p]],
 [s3, [q,p,c]], [s4, [r]]].

s0.

af(r).
```

Frågan ‘Kommer spelet alltid avslutas med ett resultat’ formulerar vi med hjälp av den logiska formeln ‘af(r)’, vilket i detta fall är falskt.

Appendix

Programkod

```
:- use_module(library(lists)).

%                              Adj    Labeling   Sate   Formula
verify(Input) :- see(Input), read(T), read(L), read(S), read(F), seen,
check(T, L, S, [], F).
```

DD1351 LAB3 2018
DIEGO LEON 961206
VIKTOR MEYER 971203

```
checkAX(T, L,[H|Tail], X) :- check(T, L, H, [], X), checkAX(T, L, Tail, X).
checkAX(T, L, [], X).

iterAdj(T, L,[H|Tail], U, X) :- check(T, L, H, U, X), iterAdj(T, L, Tail, U, X).
iterAdj(T, L, [], U, X).

% Literals
check(T, L, S, [], X) :- member([S, K], L), member(X, K).

% Negation
check(T, L, S, [], neg(X)) :- \+ check(T, L, S, [], X).

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

% Or1
check(T, L, S, [], or(F,G)) :- check(T, L, S, [], F).

% Or2
check(T, L, S, [], or(F,G)) :- check(T, L, S, [], G).

% AX
check(T, L, S, [], ax(X)) :- member([S, Adj], T), checkAX(T, L, Adj, X).

% EX
check(T, L, S, [], ex(X)) :- member([S, Adj], T), member(Nstate, Adj), check(T, L,
Nstate, [], X).

% AG 1
check(T, L, S, U, ag(X)) :- member(S, U).

% AG 2
check(T, L, S, U, ag(X)) :- \+ member(S, U), check(T, L, S, [], X), member([S,
Adj], T), iterAdj(T, L, Adj, [S|U], ag(X)).

% EG 1
check(T, L, S, U, eg(X)) :- member(S, U).

% EG 2
check(T, L, S, U, eg(X)) :- \+ member(S, U), check(T, L, S, [], X), member([S,
Adj], T), member(Node, Adj), check(T, L, Node, [S|U], eg(X)).

% EF 1
check(T, L, S, U, ef(X)) :- \+ member(S, U), check(T, L, S, [], X).

% EF 2
check(T, L, S, U, ef(X)) :- \+ member(S, U), member([S, Adj], T), member(Node,
Adj), check(T, L, Node, [S|U], ef(X)).

% AF 1
check(T, L, S, U, af(X)) :- \+ member(S, U), check(T, L, S, [], X).

% AF 2
check(T, L, S, U, af(X)) :- \+ member(S, U), member([S, Adj], T), iterAdj(T, L,
Adj, [S|U], af(X)).
```