8 December 2021 DD1351 Logik för dataloger

Rikard Johansson 010821-3570
Simon Hedström 010721-1294

Laboration 3
Modellprovning för CTL

1. Verktygsutveckling

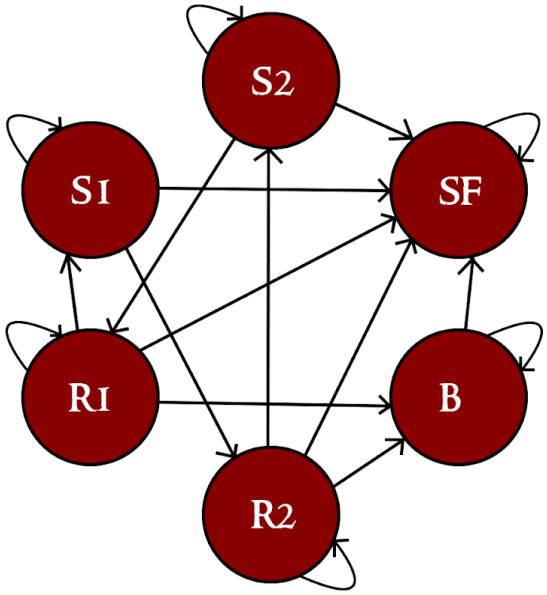
Predikat	Sann när	Falsk när
Check/5	Current state har en label och den är lika med F	Current state inte har en labeling eller current states label inte överenskommer med F
CheckAllNeighb/5	Current state har en granne och checkAllStates är sann	Saknar granne eller uppfyller ej predikatet checkAllStates
CheckAnyNeighb/5	Current state har en granne och checkAnyStates är sann	Saknar granne eller checkAnyStates är falsk
CheckAllStates/5	Alla states går igenom check	Någon av states failar vid check predikatet
CheckAnyState/5	Det finns minst en state som uppfyller check predikatet	Det inte finns något state som uppfyller check predikatet

2. Modellering

Vi har valt en hiss som modell. De atomer och tillstånd som valts finnes i listan nedan:
Atomer:
- a = alarmsignal intryckt
- p = knapp intryckt.
- v = för mycket vikt i hissen.
- h = hissen påkallad.
- n = hissen åker nedåt

Tillstånd:
- **Stilla (våning 1) = S1:**
  Atomer som gäller: a eller k eller v
- **Stilla (våning 2) = S2:**
  Atomer som gäller: a eller k eller v
- **Åker (våning 1) = R1:**
  Atomer som gäller: p eller h
- **Åker (våning 2) = R2:**
  Atomer som gäller: p eller h
- **Bromsar = B:**
  Atomer som gäller: n

- **Sitter fast = SF:**
  Atomer som gäller:
  a



3. Specificering

Systemegenskaperna som valts är:

- Hissen kan fastna när den står stilla vid våning 2:
  Sant.
- Hissen kan åka till våning X när den sitter fast:
  Falskt.

4. Verifiering

Alla testfiler har exekverats och passerat vår verktygsutveckling utan varningar.

Appendix

Programkod

```
% 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 All Neighbours-----------------
----
checkAllNeighb(T, L, S, U, F) :-
   member([S, N], T),
   checkAllStates(T, L, U, F, N).

% ----------------------Check All States----------------------
checkAllStates(_, _, _, _, []).
checkAllStates(T, L, U, F, [H|Tl]) :-
   check(T, L, H, U, F),
   checkAllStates(T, L, U, F, Tl).

% ----------------------Check Any Neighbours-----------------
----
checkAnyNeighb(T, L, S, U, F) :-
   member([S, N], T),
   checkAnyStates(T, L, U, F, N).

% ----------------------Check Any States----------------------
checkAnyStates(T, L, U, F, [H|Tl]) :-
   check(T, L, H, U, F);
   checkAnyStates(T, L, U, F, Tl).

% ----------------------Check----------------------
check(_, L, S, [], X) :-
   member([S, A], L),
   member(X, A).

% ----------------------------------------RULES-----------
---------------------------------
% ----------------------Neg----------------------
check(_, L, S, [], neg(X)) :-
   member([S, A], L),
   \+ member(X, A).
% ----------------------And----------------------
check(T, L, S, [], and(F,G)) :-
   check(T, L, S, [], F),
   check(T, L, S, [], G).
% ----------------------Or----------------------
check(T, L, S, [], or(F,G)) :-
   check(T, L, S, [], F);
   check(T, L, S, [], G).
% ----------------------AX----------------------
check(T, L, S, [], ax(F)) :-
   checkAllNeighb(T, L, S, [], F).
```

```
% ----------------------EX----------------------
check(T, L, S, [], ex(F)) :-
   checkAnyNeighb(T, L, S, [], F).
% ----------------------AG----------------------
% AG1
check(_, _, S, U, ag(_)) :-
   member(S, U).
% AG2
check(T, L, S, U, ag(F)) :-
   \+ member(S, U),
   check(T, L, S, [], F),
   checkAllNeighb(T, L, S, [S|U], ag(F)).
% ----------------------EG----------------------
% EG1
check(_, _, S, U, eg(_)) :-
   member(S, U).
% EG2
check(T, L, S, U, eg(F)) :-
   \+ member(S, U),
   check(T, L, S, [], F),
   checkAnyNeighb(T, L, S, [S|U], eg(F)).
% ----------------------EF----------------------
% EF1
check(T, L, S, U, ef(F)) :-
   \+ member(S,U),
   check(T, L, S, [], F).
% EF2
check(T, L, S, U, ef(F)) :-
   \+ member(S,U),
   checkAnyNeighb(T, L, S, [S|U], ef(F)).
% ----------------------AF----------------------
% AF1
check(T, L, S, U, af(F)) :-
   \+ member(S, U),
   check(T, L, S, [], F).
% AF2
check(T, L, S, U, af(F)) :-
   \+ member(S, U),
   checkAllNeighb(T, L, S, [S|U], af(F)).
```

Exempelmodell

[[s1,[s1,r2,sf]], [s2,[s2,r1,sf]], [r1,[r1,b,sf,s1]], [r2,[r2,b,sf,s2]], [b, [sf]], [sf,[sf]]].
[[s1, [k,o,a]], [s2, [k,o,a]], [r1, [k,h]], [r2, [k,h]], [b, [f]],
[sf, [a]]].

Specifiering

Exempel som inte håller

[[s1,[s1,r2,sf]], [s2,[s2,r1,sf]], [r1,[r1,b,sf,s1]], [r2,[r2,b,sf,s2]], [b, [sf]], [sf,[sf]]].
[[s1, [p,v,a]], [s2, [p,v,a]], [r1, [p,h]], [r2, [p,h]], [b, [n]],
[sf, [a]]].
sf.
ef(p).

Exempel som håller

```
[[s1,[s1,r2,sf]], [s2,[s2,r1,sf]], [r1,[r1,b,sf,s1]], [r2,[r2,b,sf,s2]], [b, [sf]], [sf,[sf]]].
[[s1, [p,v,a]], [s2, [p,v,a]], [r1, [p,h]], [r2, [p,h]], [b, [n]],
[sf, [a]]].
s2.
ef(n).
```