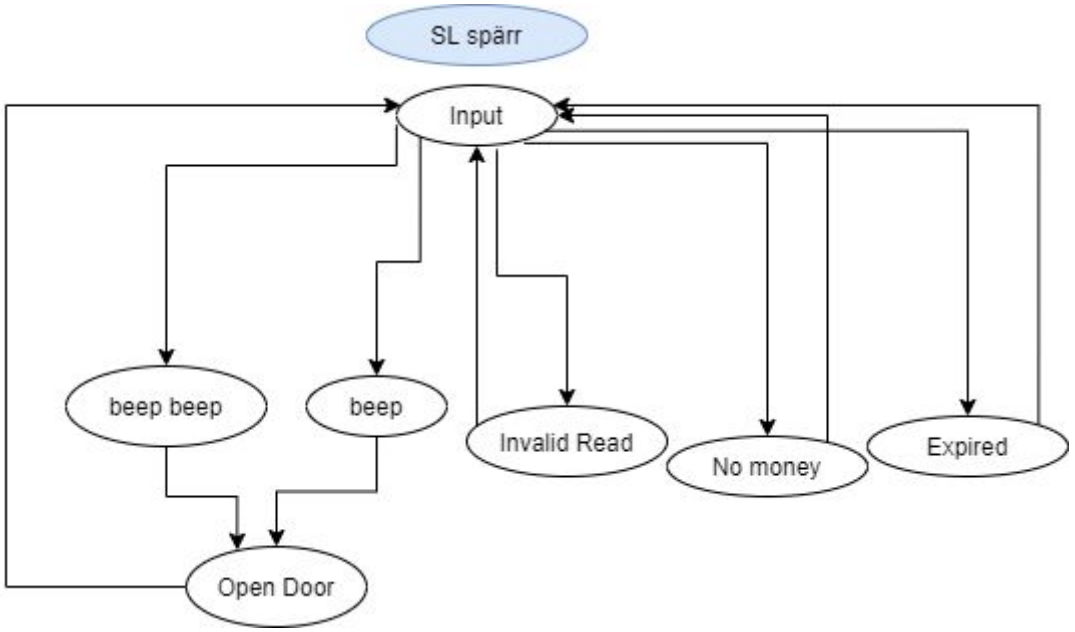
Noel Karlsson Johnasson (noelkj@kth.se)
Richard Palm (ricpal@kth.se)

Labb 3: Modellprovning för CTL

1 Modellering



1.1 Tillstånd

- Input (in) : Dörren är stängt, inget lyser, den inväntar ett kort
- Two beeps (b2) : Kortet är giltigt och det finns pengar, kortet är rabatterat
- One beep (b1) : Kortet är giltigt och det finns pengar, kortet är inte rabatterat
- Door open (do) : Dörren är öppen
- Invalid Read (ir) : Gör ett ljud och lyser rött och det står invalid read
- No Money (nm) : Gör ett ljud och lyser rött och det står att du har för lite pengar
- Expired (ex) : Gör ett ljud och lyser rött och det står att ditt kort har gått ut

1.2 Atomer

- Rabatterat kort (rk) : Håller i in, b1 och do
- Normalt kort (nk) : Håller i in, b1 och do
- Ogiltigt kort (ok) : Håller i in, ir, nm och ex

```
[[in, [b1, b2, ir, nm, ex]],
 [b2, [do]],
 [b1, [do]],
 [do, [in]],
 [ir, [in]],
 [nm, [in]],
 [ex, [in]]].

[[in, [rk, nk, ok]],
 [b2, [rk]],
 [b1, [nk]],
 [do, [rk, nk]],
 [ir, [ok]],
 [nm, [ok]],
 [ex, [ok]]].
```

2.1 Specifiering

Falskt

```
% In any branch from "input" both the negation of "ogiltigt kort"
and "normal kort" should hold. This is false because they both
hold in "input"


in.

eg(and(neg(ok),nk)).
```

Sant

```
%In all branches from the successors of input it holds that either you have a card
that is valid or invalid

in.

ax(ag(or(ok, or(nk,rk)))).
```

Predikat

Noel Karlsson Johnasson (noelkj@kth.se)
Richard Palm (ricpal@kth.se)

Namn	Är sant när
check(_, L, S, [], F)	F håller i S
check(T, L, S, [], neg(F))	F håller inte i S
check(T, L, S, [], and(F,G))	F och G håller i S
check(T, L, S, [], or(_,G))	Om G håller i S
check(T, L, S, [], or(F,_))	Om F håller i S
check(T, L, S, U, ax(F))	F ska hålla i alla efterföljare till S
check(T, L, S, U, ex(F))	F ska hålla i någon efterföljare till S
check(T, L, S, U, ag(F))	F ska hålla i S och alla grenar som kommer från S
check(T, L, S, U, eg(F))	F ska hålla i S och minst en gren från S
check(T, L, S, U, ef(F))	F ska hålla i någon state som finns i trädet som utgår från S
check(T, L, S, U, af(F))	F ska hålla i någon state i varje gren som finns i trädet som utgår från S
getInnerList([[S,Res]|_], S, Res)	Så länge det första argumentet är en lista
checkListOfStates(T, L, [Head|Tail], U ,F)	F håller i alla states i listan

Appendix

```
% 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
```

Noel Karlsson Johnasson (noelkj@kth.se)
Richard Palm (ricpal@kth.se)

```
% S - Current state
% U - Currently recorded states
% F - CTL Formula to check.
%
% Should evaluate to true if the sequence below is valid.
%
% (T,L), S |- F
% U
% To execute: consult('your_file.pl'). verify('input.txt').

%Literals
check(_, L, S, [], F) :-
    %Make sure F holds in state S
    getInnerList(L, S, List),
    member(F, List).

% Negation
check(T, L, S, [], neg(F)) :-
    %You want to make sure that F does not hold in S. You can do that
by checking that F is not a member of the state S or that check
check(T, L, S, [], F) fails
    \+ check(T, L , S, [], F).

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

% Or : check that either F or G holds
check(T, L, S, [], or(_,G)) :-
    check(T, L, S, [], G).
check(T, L, S, [], or(F,_)) :-
    check(T, L, S, [], F).

% AX F should hold in all successor to S
check(T, L, S, U, ax(F)) :-
    getInnerList(T, S, ListOfStates),
    %showList(ListOfStates),
    %Check that F holds in all states in ListOfStates
    checkListOfStates(T, L, ListOfStates, U, F).

% EX F should hold in any successor to S
```

```
check(T, L, S, U, ex(F)) :-
    getInnerList(T, S, ListOfStates),
    % Get a State from ListOfStates. Prolog will backtrack to check
next ListMember if check fails
    member(ListMember, ListOfStates),
    check(T, L, ListMember, U, F).

% AG F should hold in all states and in all branches from S
% AG1
check(_, _, S, U, ag(_)) :-
    member(S, U).

% AG2
check(T, L, S, U, ag(F)) :-
    \+ member(S, U),
    check(T, L, S, [], F),
    getInnerList(T, S, ListOfStates),
    checkListOfStates(T, L, ListOfStates, [S|U], ag(F)).

% EG F should hold in any branch from S
% EG1
check(_, _, S, U, eg(_)) :-
    member(S, U).

% EG2
check(T, L, S, U, eg(F)) :-
    \+ member(S, U),
    check(T, L, S, [], F),
    getInnerList(T, S, ListOfStates),
    % Get a State from ListOfStates. Prolog will backtrack to check
next ListMember if check fails
    member(ListMember, ListOfStates),
    check(T, L, ListMember, [S|U], eg(F)).

% EF F should hold anywhere in the tree from S
% 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),
    getInnerList(T, S, ListOfStates),
    % Get a State from ListOfStates. Prolog will backtrack to check
next ListMember if check fails
    member(ListMember, ListOfStates),
    check(T, L, ListMember, [S|U], ef(F)).

% F should hold anywhere in every branch from S
% 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),
    %check(T, L, S, [S|U], ax(af(F))).
    getInnerList(T, S, ListOfStates),
    checkListOfStates(T, L, ListOfStates, [S|U], af(F)).

%Gets the Inner list at state S
getInnerList([[S,Res]|_], S, Res) :- !.
getInnerList([_|T], S, Res) :- getInnerList(T,S,Res).

%Check that a F holds in all states of a List of states
checkListOfStates(_, _, [], _ ,_).
checkListOfStates(T, L, [Head|Tail], U ,F) :-
    check(T, L, Head, U, F), !,
    checkListOfStates(T, L, Tail, U ,F).
```

Noel Karlsson Johnasson (noelkj@kth.se)
Richard Palm (ricpal@kth.se)