DD1351 Logik för dataloger
Laboration 3: Model checker CTL

Hibatallah Belhajali

December 2022

1 Introduction

Model checker is a tool which checks whether a temporal logic formula ϕ holds in certain state s in a given model M. The task in this laboratory is to construct and implement a model checker for the rules we have in the temporal logic CTL. The purpose of this lab assignments is to be able to build simple but useful software tools, model system behavior with transition systems, specify system behavior characteristics with CTL and verify the properties with the tool that has built.

2 Model checker CTL:

In this subtask we will start from the CTL proof system in Figure 1 (next page) and write a model tester in Prolog that does a proof search from this figure.
Implementation for this model tester involves the predicate **verify** getting in data as a file, this predicate will in turn call another predicate which is called **check**, this predicate has the task of checking the validity for a given temporal logic formula. Check predicate is defined for all rules in temporal logic CTL. As some temporal logic formulas need to pass several transitions for a given node, we will need to implement two other predicates which is called **checkAllWay** this predicate has the task of checking all ways from a node. As well as the predicate **checkSomeWay** which has the task of check some of the paths, if one path gives validity then no checking of one another path needs to be performed.
In order to be able to overcome infinite loops, we have implemented two different predicates, **checkAll** This predicate will check all possible loops by passing it to the **checkAllWay** predicate, using of an empty list, the transitions that have been passed are saved and does not need to be passed again. The same applies to the second predicate **checkSome**, this predicate will check any of the subsequent loops by passing it on to the **checkSomeWay** predicate. The algorithm is run recursively and will return true if formula is correct otherwise it will return false.

2.1 A proof system for CTL

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

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

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

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

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

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

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

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

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

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

Figur 1: A proof system for CTL.

Source figure 1: Michael Huth, Mark Ryan "Logic in Computer Science" Cambridge University Press 2004 (2nd edition) ISBN 0 521 54310X.

2.1.1 The code for model checker:

```
verify(Input):-see(Input),read(T),read(L),read(S),read(F),seen,check(T,L,S,[],F),!.
% checkAllWay
checkAllWay(_,_,_,_,[]).
checkAllWay(T,L,U,F,[H|Tail]):- check(T,L,H,U,F),checkAllWay(T,L,U,F,Tail).
checkAll(T,L,S,U,F):- member([S,K],T),checkAllWay(T,L,U,F,K).
% checkSomeWay
checkSomeWay(T,L,U,F,[H|Tail]):-check(T,L,H,U,F);checkSomeWay(T,L,U,F,Tail),!.
checkSome(T,L,S,U,F) :- member([S,V],T), checkSomeWay(T,L,U,F,V),!.

% X
check(_,L,S,[],X):-member([S,A],L),member(X,A).
% not (X)
check(_,L,S,[],neg(X)):- member([S,A],L), \+member(X,A).

% AND
check(T,L,S,[],and(Y,G)):-check(T,L,S,[],Y), check(T,L,S,[],G).
% OR
check(T,L,S,[],or(Y,G)):-check(T,L,S,[],Y);check(T,L,S,[],G).

% AX
check(T,L,S,[],ax(X)):-checkAll(T,L,S,[],X).
% EX
check(T,L,S,[],ex(X)):-checkSome(T,L,S,[],X).

% AG (1)
check(_,_,S,U,ag(_)):-member(S,U).
% AG (2)
check(T,L,S,U,ag(X)):- \+member(S,U), check(T,L,S,[],X), checkAll(T,L,S,[S|U],ag(X)).

% EG (1)
check(_,_,S,U,eg(_)):- member(S,U).
% EG (2)
check(T,L,S,U,eg(X)):- \+member(S,U),check(T,L,S,[],X), checkSome(T,L,S,[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), checkSome(T,L,S,[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), checkAll(T,L,S,[S|U],af(X)).
```

2.2 Predicate table:

verify	This predicate reads the file in order to check its contents. it is true if it satisfies all the conditions, otherwise it is false.
check	Defined for each temporal logic rule and checks if which are satisfied for the specified formula then the predicate is true otherwise it is false.
checkAllWay	Checks successors of a node if it satisfies the conditions of a formula. Predicate is true if the list of all successors is empty or if those nodes satisfy the conditions of a given formula. Otherwise it is false.
checkSomeWay	Checks if any of the reachable nodes satisfy the conditions for a given condition then it is true. Otherwise, it is false if no reachable node satisfy the conditions.
checkAll	the predicate checks which other nodes can be accessed from the current node. It also calls the predicate *checkAllWay* to check all nodes. True if the conditions are satisfied
checkSome	the predicate checks which other nodes can be accessed from the current node. It also calls the *checkSomeWay* predicate to check any of the successor nodes. True if any of the successors satisfy the criteria for the specified formula.

3 Modeling:

In this subtask I am going to describe the ATM, will focus on withdrawing the money from it. To withdraw money, you must first insert your bank card, then you need to enter your pin code, if the code is wrong, the machine will give a new try for a new code. Then if you write a correct code, it will go to the next step, which is to choose the amount to withdraw, finally you can get your money.

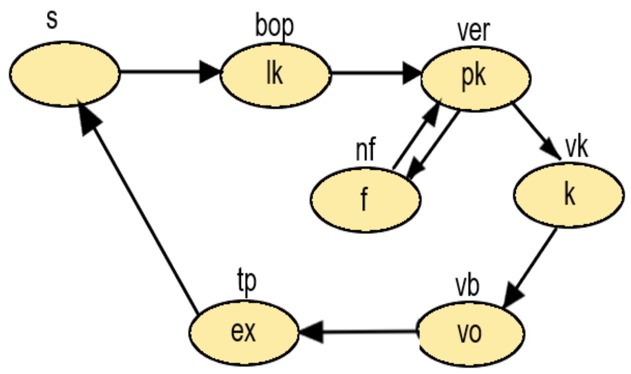
3.1 Model description

****Atoms:**
- lk: read bank card (läsa av kortet)
- pk: pin code (pinkod)
- f: wrrong (felaktigt)
- k: correct (korrekt)
- vo: choose operation(välj operation)
- ex: execute the operation.

****State:**
- s: start (still state)
- bop: start the operation (påbörja operation)
- ver: verify (verifiering)
- vk: correct code (verifiering korrekt)
- nf: new try (nytt försök)
- vb: choose amount (välja belopp)
- tp: get money (ta pegnarna)

We know that: $\mathcal{M}$ = (S, →, L)
where S = { s, bop, ver, vk, nf, vb, tp }.
And → = { (s,bop),(bop,ver),(ver,vk,nf),(nf,ver),(vk,vb),(vb,tp),(tp,s) }
And :
L(s) = {}
L(bop) = { lk}
L(ver) = { lk, pk}
L(vk) = { lk, pk, k}
L(nf) = {lk, pk, f}
L(vb) = {lk, pk, k, vo}
L(tp) = {lk, pk, k, vo, ex}

3.2 State graph:



3.3 Prolog compatible representation:

Atoms = { lk, pk, f, k, vo, ex }.
State = { s, bop, ver, vk, nf, vb, tp }
This model was translated into a suitable list structure for which the other tests to work with the proof checker implemented in prolog

The transition Function (T):
[s,[bop]],
[bop,[ver]],
[ver,[vk,nf]],
[nf,[ver]],
[vk,[vb]],
[vb,[tp]],
[tp,[s]].

The labling Function (L):
[s,[ ]],
[bop,[lk]],
[ver,[lk,pk]],
[nf,[lk,pk,f]],
[vk,[lk,pk,k]],
[vb,[lk,pk,k,vo]],
[tp,[lk,pk,k,vo,ex]],
s.

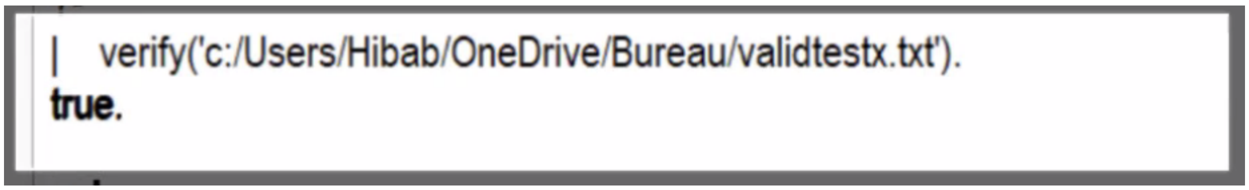
4 Specification and verification:

For the model defined above, two non-trivial system properties expressed as a CTL formula are constructed. The validity of these two system properties is tested with the model tester we have implemented, as we will create a system property that holds and one that does not, so model testers come to return as answer "false" for the incorrect property and "true" for the true one. Model tester was tested for all predefined test cases and the program has returned correct answers for all test cases.

4.1 System property that holds:

ax(ef(and(and(lk,pk),f))).

This means that for the next authorization there is always a way, where you can withdraw money if the pin code is wrong.

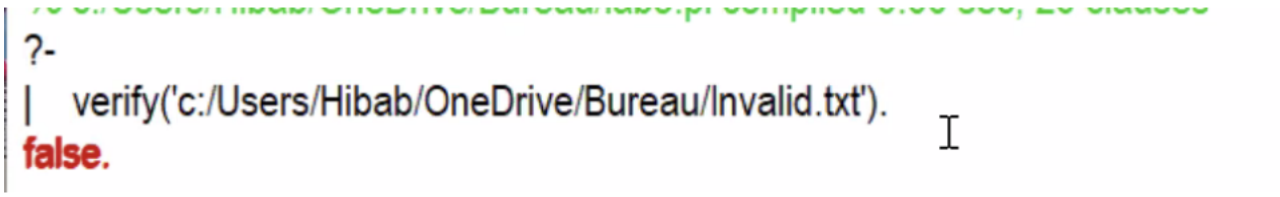


```
|   verify('c:/Users/Hibab/OneDrive/Bureau/validtestx.txt').
true.
```

4.2 System property that does not hold:

ax(ef(and(neg(and(lk,pk),k)))).

It says that for all subsequent conditions there is no way to withdraw money if the pin code is correct



```
?-
|   verify('c:/Users/Hibab/OneDrive/Bureau/Invalid.txt').
false.
```