DD1351 Logic for Computer Scientists
Alexander Järvheden 0211013693 jarvh@kth.se
Joachim Olsson 9908192876 joaco@kth.se
KTH Royal Institute of Technology
B.Sc Computer Science & Engineering
14 december 2023

DD1351 Logic for Computer Scientists

Lab 3: CTL Model Checking

Algorithm

The code and algorithm starts out by loading a given model, an initial state and formula to then proceed with the checking procedure. This initialization is done through the verify/1 predicate which reads the different transitions (T), the labeling (L), the current state (S), and a CTL formula (F) to check from an input file. Finally, the check/5 predicate is called with the given inputs, as well as an empty list U which is used to document different states that have been visited throughout the check. This predicate's purpose is to make sure that the given formula holds for a state s within the model.

From the check/5 predicates, different CTL formulas are checked based on the input format for the 5th argument. When its format is “af(F)”, “ax(F)”, or “ag(F)” they will/can invoke the check_all_future/5 predicate which checks if the formula F is true for all following transitions from a previous state S used in the check/5 predicate that invokes the check_all_future/5 predicate.

When the check/5 predicate’s fifth argument format is “ef(F)” or “eg(F)”, the exists_future/5 predicate can/will be invoked that checks if the respective formula is true for all following states from the current state. In contrast, when the check/5 predicate’s fifth argument is “ex(F)” it will invoke the predicate exists_future_ex/5 that compared to the previous exists_future/5 predicate, checks if there exists any following state from the current one in which the formula F holds without checking if the following state is in or not in the currently recorded states.

For the other check/5 calls, they work just like the normal intuitive CTL rules checking that the rules are followed and if they have nestled formulas, these are checked respectively.

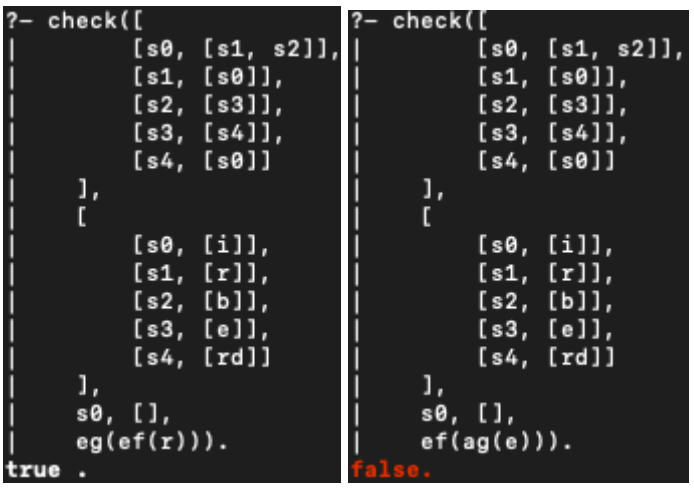
Modelling

We came up with the idea to model a computer server, which can be seen in Appendix B. This model shows the different states ranging from s1 to s4, showcasing what atom holds in which state. Our choice of atoms are as following: 1) i for idle: the system is inactive and is waiting for something to do, 2) r for running: the system is processing tasks, 3) b for busy: the system is busy executing tasks and is unable to receive more tasks, 4) e for error: an error has occurred during within the system and cannot proceed with anything, 5) rd for restart pending: the system is in need of a restart in order to return to a functioning state.

The starting state of the system is s0, which means that the system is busy. The only available choices from this state is for the system to become busy (state s2) or start running (state s1). From state s1 when the system is running, the only possible choice is for the system to become idling (s0), and if the system is busy the next available path is to s3 represented by an error occurring. Busy can therefore be interpreted as the system being extremely worked up, eventually resulting in an error. When the system reaches an error, the only next state is restart pending (s4). Finally, from this state, the only next state is for the system to become idling (s0).

```
% States are s0, s1, s2, s3, s4
    % Adjacency lists of LTS
                [
            [s0, [s1, s2]],
              [s1, [s0]],
              [s2, [s3]],
              [s3, [s4]],
              [s4, [s0]]
                ].

          % Labeling of LTS
                [
              [s0, [i]],
              [s1, [r]],
              [s2, [b]],
              [s3, [e]],
              [s4, [rd]]
                ].
```

One specification that for the state s0 holds for the model is ef(eg(r)) and one that does not is ef(ag(e)).



```
?- check([
|         [s0, [s1, s2]],
|         [s1, [s0]],
|         [s2, [s3]],
|         [s3, [s4]],
|         [s4, [s0]]
|     ],
|     [
|         [s0, [i]],
|         [s1, [r]],
|         [s2, [b]],
|         [s3, [e]],
|         [s4, [rd]]
|     ],
|     s0, [],
|     eg(ef(r))).
true .
```

```
?- check([
|         [s0, [s1, s2]],
|         [s1, [s0]],
|         [s2, [s3]],
|         [s3, [s4]],
|         [s4, [s0]]
|     ],
|     [
|         [s0, [i]],
|         [s1, [r]],
|         [s2, [b]],
|         [s3, [e]],
|         [s4, [rd]]
|     ],
|     s0, [],
|     ef(ag(e))).
false.
```

DD1351 Logic for Computer Scientists
Alexander Järvheden 0211013693 jarvh@kth.se
Joachim Olsson 9908192876 joaco@kth.se
KTH Royal Institute of Technology
B.Sc Computer Science & Engineering
14 december 2023

Implemented predicates

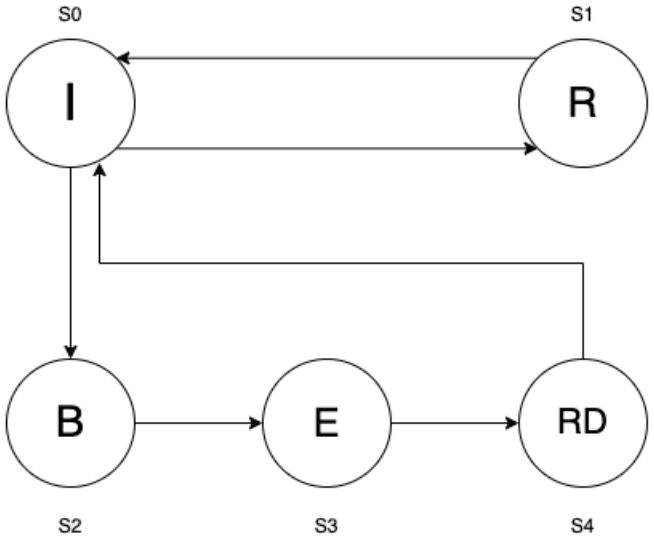
Predicate	Succeeds when
check_all_future(_, _, [], _, _)	Always when the state is empty list
check_all_future(T, L, [S1|FUTURE], U, F)	When the formula is true for all following states
exists_future(T, L, S, U, F)	If one of the formula is true for any following state
exists_future_ex(T, L, S, _, F)	True when the formula is true for a state not looking at currently checked states
check(_, L, S, [], X)	True when X is a variable for the state S
check(_, L, S, [], neg(X))	True when X is not a variable for the state S
check(T, L, S, [], and(F,G))	True when the both formulas F and G are true for state S
check(T, L, S, [], or(F,G))	True when one of the formulas F or G are true for state S
check(T, L, S, [], ax(F))	True when the formula F is true for all following states
check(T, L, S, [], ex(F))	True if the formula F is true for any following state of S
check(_, _, S, U, ag(_))	True if S is a member of U
check(T, L, S, U, ag(F))	True when S is not a member of U and the formula F holds for both the state S and AG(F) holds for all following states from S
check(_, _, S, U, eg(_))	True if S is a member of U
check(T, L, S, U, eg(F))	True if S is not a member of U and the formula F holds for the state S and EG(F) holds for any following state from S
check(T, L, S, U, ef(F))	True when S is not a member of U and either if the formula F holds for the state S or if EF(F) holds for any following state from S
check(T, L, S, U, af(F))	True when S is not a member of U and either the formula F holds for the state S or if AF(F) holds for all following states from S

Appendix A

```
% Written by Alexander Järvheden & Joachim Olsson
% 14 December 2023
% DD1351 Logic for Computer Scientists - LAB 3
% COPYRIGHT

% 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.

% Check all future states
check_all_future(_, _, [], _, _).
check_all_future(T, L, [S1|FUTURE], U, F) :- check(T, L, S1, U,
F), check_all_future(T, L, FUTURE, U, F).

% Check if theres one true existing state
exists_future(T, L, S, U, F):- member([S, TRANS], T),
member(SPRIM, TRANS), check(T, L, SPRIM, [S|U], F).

% Used for EX
exists_future_ex(T, L, S, _, F):- member([S, TRANS], T),
member(SPRIM, TRANS), check(T, L, SPRIM, [], F).

% Literals
check(_, L, S, [], X) :- member([S, VARIABLES], L), member(X,
VARIABLES).
check(_, L, S, [], neg(X)) :- member([S, VARIABLES], L), \+
member(X, VARIABLES).

% And
che
ck(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 - i alla nästa tillstånd gäller formeln.
check(T, L, S, [], ax(F)) :- member([S, TRANS], T),
check_all_future(T, L, TRANS, [], F).

% EX - i något nästa tillstånd gäller formeln.
check(T, L, S, [], ex(F)) :- exists_future_ex(T, L, S, [], F).

% AG
check(_, _, S, U, ag(_)) :- member(S, U).
check(T, L, S, U, ag(F)) :- \+ member(S, U), check(T, L, S, [],
F), member([S, TRANS], T), check_all_future(T, L, TRANS, [S|U],
ag(F)).

% EG
check(_, _, S, U, eg(_)):- member(S, U).
check(T, L, S, U, eg(F)) :- \+ member(S, U), check(T, L, S, [],
F), exists_future(T, L, S, [S|U], eg(F)).

% EF
check(T, L, S, U, ef(F)) :- \+ member(S, U), (check(T, L, S, [],
F); exists_future(T, L, S, U, ef(F))).

% AF
check(T, L, S, U, af(F)) :- \+ member(S, U), (check(T, L, S, [],
F); member([S, TRANS], T), check_all_future(T, L, TRANS, [S|U],
af(F))).
```

Appendix B



I = idle, R = running, B = busy, E = error, RD = restart pending

Appendix C - Proof system for CTL

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

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

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

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

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

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

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

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

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

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