CTL Proof-system implemented in Prolog

;

Group Lumon

Ludwig Berglind, Simon Severinsson

Modelling

A very basic, scaled-off web shop was chosen for the model design where a user can browse between products, add products to their cart, go to checkout, and completing a purchase.

The model M has the following states: s0: Start (A customer lands on the website) s1: Browsing s2: Shopping cart s3: Checkout s4: Completed purchase

The model has the following transitions: s0 -> s1: Starting to browse s1 -> s2: Add product to cart s2 -> s1: Continue browsing s2 -> s3: Go to checkout s3 -> s4: Complete purchase s4 -> s0: Return to home page after finished purchase

The model consists of the following atoms:

i: internet access c: credit card details p: Payment is processed l: Log in to the website

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

[[s0, [i]], [s1, [i]], [s2, [i,l]], [s3, [i,l,c]], [s4, [i,l,c, p]]].

s1.

ag(ef(c)).

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

[[s0, [i]], [s1, [i]], [s2, [i,l]], [s3, [i,l,c]], [s4, [i,l,c, p]]].

s1.

ef(and(neg(l), p)).

The model simplifies a basic web shop to a sequence of steps a customer can partake in. Each state represents a stage in the purchasing process, and the transitions show possible ways to navigate between these stages.

Specification: Valid: AG(s1 -> EF(card details)) If the system is in stage "Browsing", there always exists one path (EF) where card details are necessary.

Invalid: EF(¬logged in ˆ payment is processed). There exists a path where a user is not logged in (¬logged in) and a payment is being processed at the same time.

Predicates

Predikat	Sant	Falskt
assertTransitions\1	When all transitions have been asserted as facts	If the formatting of the transitions is incorrect
assertTransitionsHelper\1	When the transitions from one node have been asserted	If the formatting of the list is incorrect
assertTransition\2	When all transitions from a node have been asserted	Då beviset inte stämmer (validProof är false), eller inläsning av misslyckas
assertStates\1	When all states have been asserted	If the formatting of the states is incorrect
assertState\1	When a state has been asserted	If the formatting of a state is incorrect
checkAllPaths\3	When all paths from the current state satisfies the formula	When one or more paths from the current state dont satsify
checkAllPathsH\3	When the current path being checked satisfies the formula	When the current path being checked does not satisfy the formula
checkExistsPath\3	When one of the paths from the current state satsify the formula	When none of the paths from the current state satisfy the formula
checkExistsPathH\3	When the current path being checked satisfies the formula	When the current path being checked does not satisfy the formula
check\3	When the current state satisfies the formula	When the current state does not satisfy the formula

Predikat	Sant	Falskt
stateContains\2	When the current state satisfies the formula (formula will always be an atom)	When the current state does not satisfy the formula
abolishAll\0	Always	Never

Appendix A - Code

```
verify(InputFileName) :-
    abolishAll(),
    see(InputFileName),
    read(Adjacencies), read(States), read(State), read(Formula),
    seen,
    assertTransitions(Adjacencies),
    assertStates(States),
    check(State, [], Formula), !,
    abolishAll().

assertTransitions([]).

assertTransitions([Head | Tail]) :-
    assertTransitionsHelper(Head),
    assertTransitions(Tail).

assertTransitionsHelper([Node | [To | _]]) :-
    assertTransition(Node, To).

assertTransition(_, []).
assertTransition(Node, [Head | Tail]) :-
    assertz(transition(Node, Head)),
    assertTransition(Node, Tail).

assertStates([]).
assertStates([Head | Tail]) :-
    assertState(Head),
    assertStates(Tail).

assertState([Node | [State | _]]) :-
    assertz(state(Node, State)).

checkAllPathsH([], _, _).

checkAllPaths(State, U, Formula) :-
    findall(Next, transition(State, Next), L),
    checkAllPathsH(L, U, Formula).

checkAllPathsH([Head | Tail], [], Formula) :-
    check(Head, [], Formula),
    checkAllPathsH(Tail, [], Formula).

checkAllPathsH([Head | Tail], U, Formula) :-
    check(Head, U, Formula),
```

```
    checkAllPathsH(Tail, [Head| U], Formula).

checkExistsPath(State, [], Formula) :-
    findall(Next, transition(State, Next), L),
    checkExistsPathH(L, [], Formula).

checkExistsPath(State, U, Formula) :-
    findall(Next, transition(State, Next), L),
    checkExistsPathH(L, U, Formula).


checkExistsPathH([], _, _) :- fail.

checkExistsPathH([Head | Tail], U, Formula) :-
    check(Head, U, Formula) ;
    checkExistsPathH(Tail, [Head | U], Formula).

% Atom
check(State, _, Formula) :-
    stateContains(State, Formula).

% Neg
check(State, _, neg(Formula)) :-
    \+ stateContains(State, Formula).

% And
check(State, _, and(F, G)) :-
    check(State, [], F), check(State, [], G).

% Or
check(State, _, or(F, G)) :-
    check(State, [], F) ; check(State, [], G).

% AX
check(State, [], ax(Formula)) :-
    checkAllPaths(State, [], Formula).

% EX
check(State, _, ex(Formula)) :-
    checkExistsPath(State, [], Formula).

% AG
check(State, U, ag(_)) :-
    member(State, U).

check(State, U, ag(Formula)) :-
```

```
    \+ member(State, U),
    check(State, [], Formula),
    checkAllPaths(State, [State | U], ag(Formula)).

% EG
check(State, U, eg(_)) :-
    member(State, U).
check(State, U, eg(Formula)) :-
    \+ member(State, U),
    check(State, [], Formula),
    checkExistsPath(State, [State | U], eg(Formula)).

% EF
check(State, U, ef(Formula)) :-
    \+ member(State, U),
    check(State, [], Formula).

check(State, U, ef(Formula)) :-
    \+ member(State, U),
    checkExistsPath(State, [State | U], ef(Formula)).

% AF
check(State, U, af(Formula)) :-
    \+ member(State, U),
    check(State, [], Formula).
check(State, U, af(Formula)) :-
    \+ member(State, U),
    checkAllPaths(State, [State | U], af(Formula)).

stateContains(Node, Formula) :-
    state(Node, L),
    member(Formula, L).

abolishAll() :- abolish(transition/2), abolish(state/2).
```