

Lab 3 - Modellprovning för CTL

Hiba Qutbuddin Habib

Ayah Babiker

December 2022

Introduktion

Uppgiften i denna laborationen är att konstruera samt implementera en modellprovare för reglerna i temporallogiken CTL. Modellprovare är då ett verktyg som kontrollerar om en temporallogik formel ϕ gäller i ett visst tillstånd s i en given modell M .

Beskrivning av algoritmen

Implementering för denna modellprovare går ut på att predikaten *verify* får in data som en fil, denna predikaten kommer i sin tur att anropa en annan predikat som kallas för *check*, denna predikaten har i uppgift att kontrollera validiteten för en given temporallogik formel. *Check* predikat defineras för samtliga regler i temporallogik CTL. Då vissa temporallogiksa formler behöver passera flera övergångar för en given nod, kommer vi behöva implementera två andra predikat som kallas för *checking_all_ways* denna predikat har i uppgift att kontrollera alla vägar från en nod, samt predikaten *checking_som_ways* som har i uppgift att kontrollera några av vägarna, ifall en väg ge validitet så ingen kontroll av en annan väg behöver utföras.

För att kunna undervika oändliga loopar så har vi implementerat två olika predikat, *checking_All* denna predikat kommer att kontrollera alla möjliga slingor genom att den skickas vidare till predikaten *checking_all_ways*, med hjälp av en tom lista sparas de övergångarna som har passerats så att en och samma övergång behövs ej passeras igen. Detsamma gäller den andra predikaten *checking_some*, dock denna predikaten kommer att kontrollera någon av de efterföljare slingorna genom att den skickas vidare till predikaten *checking_some_ways*. Algoritmen körs rekursivt och kommer att returnera true om formel är korrekt annars returnerar den false om formel är ej korrekt.

Predikattabell


Predikat	Beskrivning	Validitet
verify	Denna predikat läser in filen för att sedan kontrollera innehåller med algoritmen .	Denna predikat är sant om formel uppfylla samtliga villkor annars är den falskt.
checking_all_ways/5	Kontrollerar samtliga efterföljare för en node ifall de uppfyller villkor för en formel.	Predikaten är sann om listan för samtliga efterföljare är tom eller om de noder uppfyller villkor för en angiven formel. Denna är inte sant om någon nåbar noden inte uppfyller villkor för en formel .
cheking_some_way/5	Kontrollerar om någon av de nåbara noder uppfyller villkor för en formel.	Predikaten är sann om någon av de nåbara noder uppfyller villkor för en angiven formel, annars är den falsk om ingen nåbar nod uppfyll villkor.
cheking_All/5	Predikaten kollar vilka andra noder man kan gå till från den nuvarande noden. Predikaten kallar även den predikaten cheking_all_ways, för att kontrollera samtliga noder	Predikaten är sann om villkorna är uppfyllda annars är den falskt.
cheking_some/5	Predikaten kollar vilka andra noder man kan gå till från den nuvarande noden. Predikaten kallar även den predikaten cheking_some_ways, för att kontrollera någon av de efterföljare noder	Predikaten är sann om någon efterföljare uppfyller villkor för en angiven formel annars är det falskt.
check/5	Definieras för varje temporallogik regel och kontrollera om villkor uppfylls för en angiven formel. 	Predikaten är sann om den angivna formel uppfyller regler annars är den falskt.

Figure 1: Beskrivning av predikat samt deras validitet

Modellering

I denna delen tänkte vi beskriva *Selectaautomat*, alltså modellen när man ska köpa någon vara från en *Selectaautomat*. Modellen kommer alltid att starta

från stilla tillstånd, för att sedan man ska sätta in bankkortet för avläsning. När avläsning är färdig då väljer man vara, man anger alltså nummer på den produkten man önskar ha. Sedan kommer man behöva trycka in pinkod för bankkortet, om pinkoden är godkänd då kommer vara att falla ner sedan är man redo att ta den, om koden är felaktig då kommer man kunna ha möjlighet till ett nytt försök.

Modellbeskrivning

Atomer: ak(avläsa kortet), vap(val av produkt), p(pinkod), f(felaktig), g(godkänd), tap(tag produkt).

Tillstånd: s(start), pak(påbörja köpet), vv(val av vara), ver(verifiering), verk(verifiering är klar), fi(försök igen), tv(tag vara).

$M = (S, ->, L)$, där:

$S = \{s, pak, , vv, ver, verk, fi, tv\}$.

$-> = \{(s, pak), (pak, vv), (vv, ver), (ver, verk, fi), (fi, ver), (verk, tv), (tv, s)\}$.

$L(s) = \{\}$

$L(pak) = \{ak\}$

$L(vv) = \{ak, vap\}$

$L(ver) = \{ak, p\}$

$L(verk) = \{ak, p, g\}$

$L(fi) = \{ak, p, f\}$

$L(tv) = \{ak, p, g, tap\}$

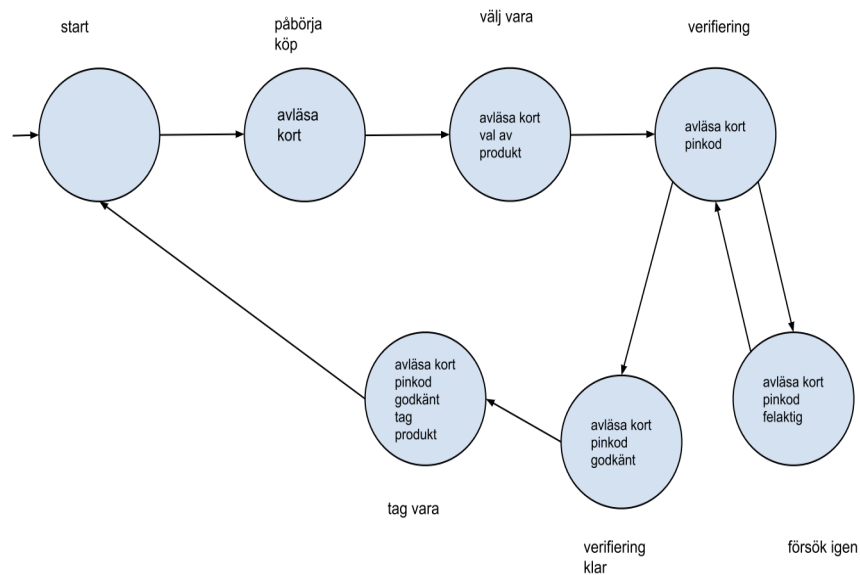


Figure 2: Tillståndsgraf

Prolog-kompatibel representation av modellen

Vi startar alltid i stillatillstånd.

Atomer = {ak, vap, p, f, g, tap}.

Tillstånd = {s, pak, , vv, ver, verk, fi, tv}.

```
[[s,[pak]],  
 [pak,[vv]],  
 [vv,[ver]],  
 [ver,[verk,fi]],  
 [fi,[ver]],  
 [verk,[tv]],  
 [tv,[s]]].
```

```
[[s,[]],  
 [pak,[ak]],  
 [vv,[ak,vap]],  
 [ver,[ak,p]],  
 [verk,[ak,p,g]],  
 [fi,[ak,p,f]],  
 [tv,[ak,p,g,tap]]].
```

s.

Specifiering

För den definierade modellen ovan konstrueras två icke-triviala systemegenskaper uttryckta som CTL-formel. Validiteten för dessa två systemegenskaperna testas med den modellprovare som vi har implementerat, då vi kommer att skapa en systemegenskap som håller och en som inte håller, så modellprovare kommer att returnera som svara "false" för den felaktig egenskapen samt "true" för denna sanna egenskapen.

Modellprovare testades för alla fördefinierade testfall och programmet har returnerat korrekta svar för samtliga testfall.

Systemegenskapen som *Håller*

För alla nästa tillstånd existerar det en väg för att tag ett vara om pin kod är felaktig.

$ax(ef(\text{and}(\text{and}(ak,p),f)))$.

```
?- verify('/Users/hibaqutbuddinhabib/desktop/validtestx.txt').  
true.
```

Figure 3: validtestx.txt

Systemegenskapen som *Intehåller*

För alla nästa tillstånd existerar inte en väg för att tag ett vara om pin kod är godkänt.

$\text{ax}(\text{ef}(\text{and}(\text{neg}(\text{and}(\text{ak}, \text{p}), \text{g}))))).$

```
?- verify('/Users/hibaqutbuddinhabib/desktop/invalidtest.txt').
false.
```

Figure 4: invalidtext.txt

Appendix

Regler

$$\begin{array}{c}
 \begin{array}{cc}
 p \frac{-}{\mathcal{M}, s \vdash []} p \in L(s) & \neg p \frac{-}{\mathcal{M}, s \vdash []} p \notin L(s) \\
 \wedge \frac{\mathcal{M}, s \vdash [] \phi \quad \mathcal{M}, s \vdash [] \psi}{\mathcal{M}, s \vdash [] \phi \wedge \psi} \\
 \vee_1 \frac{\mathcal{M}, s \vdash [] \phi}{\mathcal{M}, s \vdash [] \phi \vee \psi} & \vee_2 \frac{\mathcal{M}, s \vdash [] \psi}{\mathcal{M}, s \vdash [] \phi \vee \psi} \\
 \text{AX} \frac{\mathcal{M}, s_1 \vdash [] \phi \quad \dots \quad \mathcal{M}, s_n \vdash [] \phi}{\mathcal{M}, s \vdash [] \text{AX } \phi} \\
 \text{AG}_1 \frac{-}{\mathcal{M}, s \vdash_U \text{AG } \phi} s \in U & \text{AF}_1 \frac{\mathcal{M}, s \vdash [] \phi}{\mathcal{M}, s \vdash_U \text{AF } \phi} s \notin U \\
 \text{AG}_2 \frac{\mathcal{M}, s \vdash [] \phi \quad \mathcal{M}, s_1 \vdash_{U,s} \text{AG } \phi \quad \dots \quad \mathcal{M}, s_n \vdash_{U,s} \text{AG } \phi}{\mathcal{M}, s \vdash_U \text{AG } \phi} s \notin U \\
 \text{AF}_2 \frac{\mathcal{M}, s_1 \vdash_{U,s} \text{AF } \phi \quad \dots \quad \mathcal{M}, s_n \vdash_{U,s} \text{AF } \phi}{\mathcal{M}, s \vdash_U \text{AF } \phi} s \notin U \\
 \text{EX} \frac{\mathcal{M}, s' \vdash [] \phi}{\mathcal{M}, s \vdash [] \text{EX } \phi} & \text{EG}_1 \frac{-}{\mathcal{M}, s \vdash_U \text{EG } \phi} s \in U \\
 \text{EG}_2 \frac{\mathcal{M}, s \vdash [] \phi \quad \mathcal{M}, s' \vdash_{U,s} \text{EG } \phi}{\mathcal{M}, s \vdash_U \text{EG } \phi} s \notin U \\
 \text{EF}_1 \frac{\mathcal{M}, s \vdash [] \phi}{\mathcal{M}, s \vdash_U \text{EF } \phi} s \notin U & \text{EF}_2 \frac{\mathcal{M}, s' \vdash_{U,s} \text{EF } \phi}{\mathcal{M}, s \vdash_U \text{EF } \phi} s \notin U
 \end{array}
 \end{array}$$

Figure 5: Formler ovan hanterades i algotitemen

Koden för modellprovare

```
:- disjoint check/5.

verify(Input):-see(Input),read(T),read(L),read(S),read(F),
seen,check(T,L,S,[],F),!.

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

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

checking_All(T,L,S,U,F):- member([S,K],T),
checking_all_ways(T,L,U,F,K).

% check all possible ways (A)
checking_all_ways(_,_,_,_,[]).
checking_all_ways(T,L,U,F,[H|Tail]):- check(T,L,H,U,F),
checking_all_ways(T,L,U,F,Tail).

% check some routes(E)
checking_some_ways(T,L,U,F,[H|Tail]):-check(T,L,H,U,F);
checking_some_ways(T,L,U,F,Tail),!.

checking_some(T,L,S,U,F) :- member([S,V],T),
checking_some_ways(T,L,U,F,V),!.

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

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

% check ax(X)
check(T,L,S,[],ax(X)):-checking_All(T,L,S,[],X).

% check ex(X)
check(T,L,S,[],ex(X)):-checking_some(T,L,S,[],X).

% check ag(X) basfall (1)
check(_,_,S,U,ag(_)):-member(S,U).

% check ag(X) (2)
```

```

check(T,L,S,U,ag(X)):- \+member(S,U), check(T,L,S,[],X),
checking_All(T,L,S,[S|U],ag(X)).

% check af(X) (1)
check(T,L,S,U,af(X)):- \+member(S,U),check(T,L,S,[],X).

% check af(X) (2)
check(T,L,S,U,af(X)):- \+member(S,U),
checking_All(T,L,S,[S|U],af(X)).

% check eg(X) basfall (1)
check(_,-,S,U,eg(_)):- member(S,U).

% check eg(X) (2)
check(T,L,S,U,eg(X)):- \+member(S,U),check(T,L,S,[],X),
checking_some(T,L,S,[S|U],eg(X)).

% check ef(X) (1)
check(T,L,S,U,ef(X)):- \+member(S,U), check(T,L,S,[],X).

% check ef(X) (2)
check(T,L,S,U,ef(X)):- \+member(S,U),
checking_some(T,L,S,[S|U],ef(X)).

```