Laboration 3: Modellprovning för CTL

Adeel Hussain adeelh@kth.se, Philip Tonaczew tonaczew@kth.se

11 december 2020

1 Inledning

Konstruera och implementera en modellprovare som kontrollerar om en temporallogisk formel ϕ gäller i ett visst tillstånd s i en given modell M för en delmängd av reglerna i temporallogiken CTL.

2 Generell beskrivning av algoritmen

Algoritmen får in data via predikatet $verify$ som tar in en fil innehållandes modellen, starttillståndet och en formel. Detta skickas sedan vidare till predikatet *check*, som ska kontrollera och verifiera om den angivna formeln är korrekt konstruerad, för att ta sig igenom modellens samtliga tillstånd och övergångar. För att kontrollera det har *check* definierats för respektive formel (anges i ett appendix) som behöver godkännas enligt formelns villkor, för att kunna kontrollera eventuella resterande delar av formeln. För de formler som kräver flera övergångar till olika noder görs det med hjälp av predikatet $checkALL$ eller $checkSOME$, beroende på vad villkoret för formeln avser. För att undvika att algoritmen hamnar i en oändlig loop och inte terminerar, vilket kan inträffa vid tillstånd som har *slingor*, används från början en tom lista som fylls på med de tillstånd som redan har kontrollerats vid en övergång, för att vid nästa övergång kunna kontrollera om tillståndet redan kontrollerats och därmed terminera. Algoritmen arbetar rekursivt så länge varje villkor för respektive formel är uppfyllt fram tills hela formeln är genomförd och svarar *true* om den är korrekt annars *false*.

3 Lista över predikat

Predikat	Beskrivning	Sant	Falsk
verify/1	Läser in en textfil vars innehåll ska kontrolleras av algoritmen .	När innehållande formel uppfyller samtliga villkor.	När innehållande formel inte uppfyller något villkor.
check/5	Testar om villkoren uppfylls för berörd formel.	När villkoren för berörd formel uppfylls.	När villkoren för berörd formel inte uppfylls.
checkALL/5	Kontrollerar att samtliga noder i listan för nåbara noder uppfyller villkoren för angiven formel.	När listan för nåbara noder är tom samt/eller då samtliga noder i listan uppfyller villkoren för angiven formel.	När villkoren för någon/inte alla nod(er) i listan för nåbara noder inte uppfylls för angiven formel.
checkSOME/5	Kontrollerar att någon nod i listan för nåbara noder uppfyller villkoren för angiven formel.	När någon nod i listan för nåbara noder uppfyller villkoren för formel.	När listan för nåbara noder är tom eller när någon nod i listan inte uppfyller villkoren för angiven formel.

4 Modellering

Vår modell beskriver en variant av en bankomat, där det är tänkt att kunna göra en insättning eller ett uttag. Modellen startar alltid i tillståndet start utan atomer och efterfrågar en PIN-kod som kontrollerar om inmatningen är korrekt eller inte, vid fel inmatning är det möjligt att försöka igen. Korrekt inmatning övergår till en meny för val om att göra en insättning eller ett uttag och därefter genomförs aktiviteten. Det är möjligt att efter något val i menyn återgå tillbaka till menyn om exempelvis fel knappval gjorts eller återgå tillbaka till start från menyn eller, inmatning av PIN-kod om användaren av någon anledning vill få tillbaka kortet utan aktivitet. Modellen visas som tillståndsgraf i figur 1.

4.1 Modellen

Atomer: p (pin), f (felaktig), g (godkänd), k1 (knapp1), k2 (knapp2), be (belopp).
Tillstånd: st (start), ver (verifiering), fi (försök igen), me (meny), ut (uttag), in (insättning), sa (saldo).

M = $(S, \rightarrow, L)$, där

$S = \{st, ver, fi, me, ut, in, sa\}$
$\rightarrow = \{(st, ver), (ver, st, fi, me), (fi, ver), (me, ut, in, st), (ut, me, sa), (in, me, sa), (sa, st)\}$
$L(st) = \{\}$
$L(ver) = \{p\}$
$L(fi) = \{p, f\}$
$L(me) = \{p, g\}$
$L(ut) = \{p, g, k1\}$
$L(in) = \{p, g, k2\}$
$L(sa) = \{p, g, be\}$
$Atomer = \{p, f, g, k1, k2, be\}$

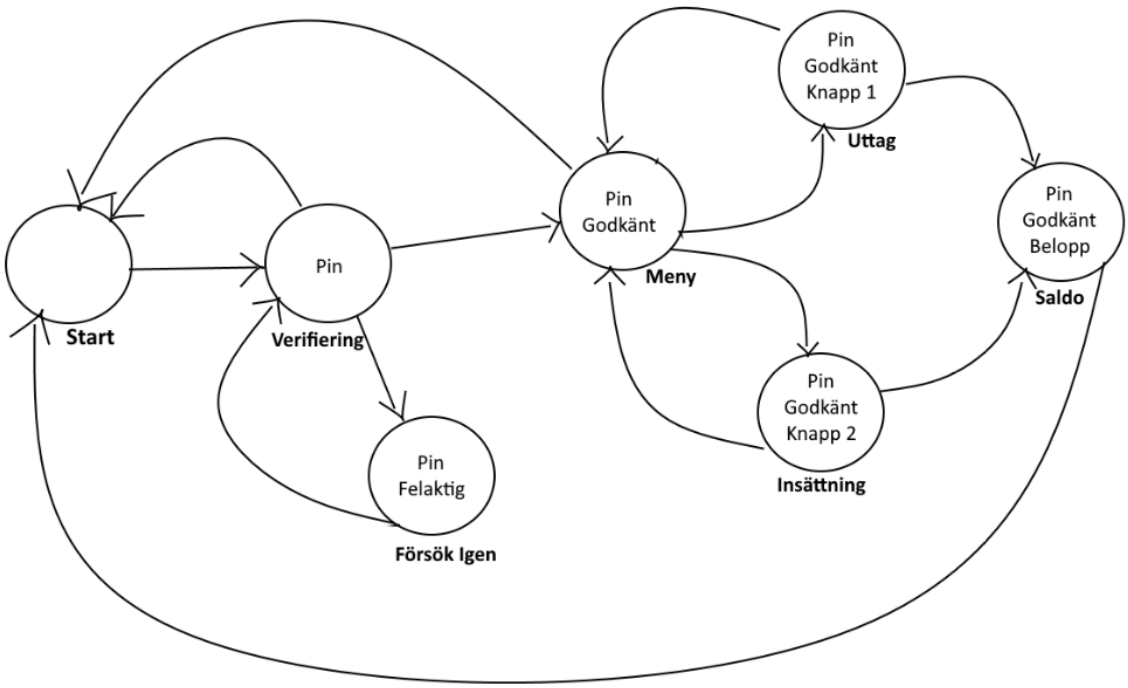

Figure 1: Tillståndsgraf

4.2 Prolog-kompatibel representation av modellen

Modellen startar alltid i tillståndet start

```
% Atomer = {p, f, g, k1, k2, be}
% Tillstånd: {st, ver, fi, me, ut, in, sa}

[[st, [ver]],
[ver, [st, fi, me]],
[fi, [ver]],
[me, [ut, in, st]],
[ut, [me, sa]],
[in, [me, sa]],
[sa, [st]]].

[[st, []],
[ver, [p]],
[fi, [p,f]],
[me, [p,g]],
[ut, [p,g,k1]],
[in, [p,g,k2]],
[sa, [p,g,be]]].

st.
```

5 Specificering & Verifiering

Följande två systemegenskaper uttryckta i CTL-formler relaterad till vår modell, en som håller och en som inte håller.
Systemegenskaperna i exemplen höll respektive höll inte som precis som förväntat samt att samtliga fördefinierade testfall gav korrekta resultat.

5.1 Håller

(validex.txt)
För alla nästa tillstånd existerar det en väg som gör det möjligt att välja mellan att göra en insättning eller ett uttag.

```
ax(ef(or(and(and(p,g),k1),and(and(p,g),k2)))).
```

```
2 ?- verify('tests/validex.txt').
true.
```

5.2 Inte håller

(invalidex.txt)
För alla nästa tillstånd existerar det en väg som gör det möjligt att ta in/ut ett belopp med en felaktig PIN-kod.

```
ax(ef(and(and(p,g),f))).
```

```
3 ?- verify('tests/invalidex.txt').
false.
```

Appendix

Formler

Samtliga formler hanteras av algoritmen:

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

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

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

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

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

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

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

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

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

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

Koden för algoritmen

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

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

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

%% AND
check(T, L, S, [], and(X1,X2)):-
    check(T, L, S, [], X1),
    check(T, L, S, [], X2).

%% OR
check(T, L, S, [], or(X1,X2)):-
    check(T, L, S, [], X1);
    check(T, L, S, [], X2).

%% AX - Alla nästa tillstånd gäller fi
check(T, L, S, [], ax(X)):-
    member([S,Q], T),
    checkALL(T, L, Q, [], X).

%% EX - I något nästa tillstånd gäller fi
check(T, L, S, [], ex(X)):-
    member([S,Q], T),
    checkSOME(T, L, Q, [], X).

%% AG - För alla utvecklingar gäller fi
% AG1 - basfall
check(_, _, S, U, ag(_)):-
    member(S, U).
%AG2
check(T, L, S, U, ag(X)):-
    \+ member(S, U),
    check(T, L, S, [], X),
    member([S,Q], T),
    checkALL(T, L, Q, [S|U], ag(X)).
```

```
%% EG - Det finns en väg där fi alltid gäller
%EG1 - basfall
check(_, _, S, U, eg(_)):-
    member(S, U).

% EG2
check(T, L, S, U, eg(X)):-
    \+ member(S, U),
    check(T, L, S, [], X),
    member([S,Q], T),
    checkSOME(T, L, Q, [S|U], eg(X)).

%% EF - Det finns en väg där fi gäller så småningom
%EF1 - basfall
check(T, L, S, U, ef(X)):-
    \+ member(S, U),
    check(T, L, S, [], X).

%EF2
check(T, L, S, U, ef(X)):-
    \+ member(S, U),
    member([S,Q], T),
    checkSOME(T, L, Q, [S|U], ef(X)).

%%AF - Över alla vägar gäller fi så småningom
% AF1 - basfall
check(T, L, S, U, af(X)):-
    \+ member(S, U),
    check(T, L, S, [], X).

% AF2
check(T, L, S, U, af(X)):-
    \+ member(S, U),
    member([S,Q], T),
    checkALL(T, L, Q, [S|U], af(X)).

%% Predikat för att kolla alla vägar 'A'
checkALL(_, _, [], _, _).
checkALL(T, L, [Q1|Tail], U, X):-
    check(T, L, Q1, U, X),
    checkALL(T, L, Tail, U, X).

%% Predikat för att kolla någon väg 'E'
checkSOME(T, L, [Q1|Tail], U, X):-
    check(T, L, Q1, U, X);
    checkSOME(T, L, Tail, U, X).
```