Joseph Karroum
Tomas Weldetinsae

LABB 3 : Rapport

Resultat

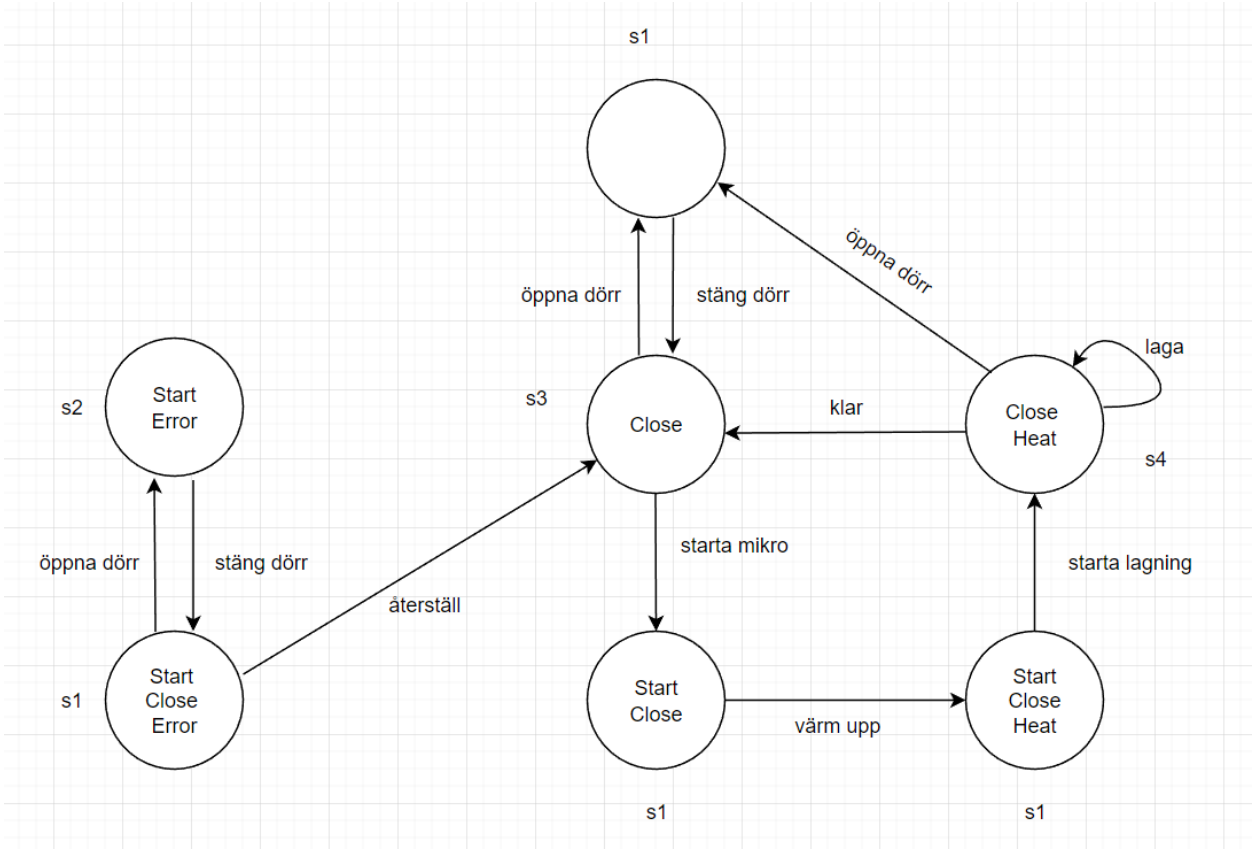
Efter att ha skrivit programkoden för att skapa en modell-provare som är baserad på bevissystemet för CTL, har vi kommit fram till ett program som givet en modell M, gör bevissökning i CTL bevissystemet och returnerar ett resultat på verifikation om en viss CTL-formel som tas in som input.

Som uppgiften lydde, så ska vi definiera ett exempel på en modell.

```
[[s1, [s3]],
 [s2, [s5]],
 [s3, [s1, s6]],
 [s4, [s1, s3, s4]],
 [s5, [s2, s3]],
 [s6, [s7]],
 [s7, [s4]]].
```

```
[[s1, []],
 [s2, [start, error]],
 [s3, [close]],
 [s4, [close, heat]],
 [s5, [start, close, error]],
 [s6, [start, close]],
 [s7, [start, close, heat]]].
```

Här nedan har vi gjort en figur på modellen. Den bygger på modellering av mikrovågsugn.



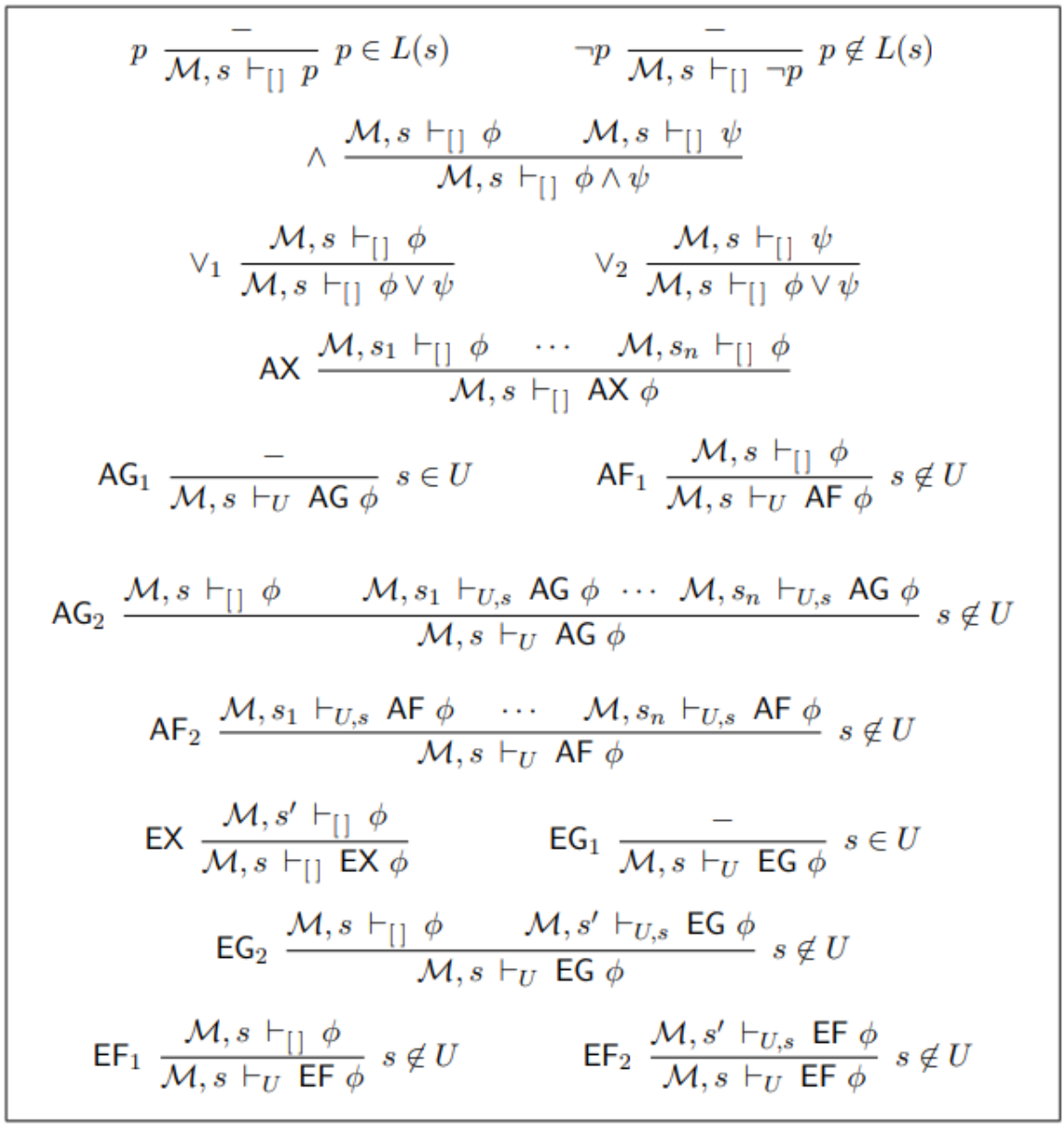
Tabell över predikaten i programmet

Modelprövaren använder sig av rekursion och back-tracking för att validera en viss CTL-formel som läses in tillsammans med en modell. Vi har skrivit ner alla predikat som används för att validera CTL-formler i modelprövaren.

Namn på predikat	*När det är sant/falskt?*
verify	Det blir sant när CTL-formeln (som betecknas som F efter inläsning i modelprövaren) är validerbar med hjälp av predikat “check”
check	Det blir sant när aktuell CTL-formel är sann, dvs om en viss CTL-formel gäller i ett visst tillstånd blir predikatet sant. Annars falskt.
check_for_some	Det blir sant när någon av grenarna som undersöks är sann (eller fler än en, men en räcker), annars blir den falsk. Undersökningen sker genom att binda en av predikaten som vi har definierat med “check”.
check_for_all	Det blir sant när alla grenar som undersöks är sanna (inget mindre!), annars blir den falsk. Undersökningen sker på samma sätt som för check_for_some.

Formlerna som formaliserar beteendeegenskaperna

(Hämtad ur labb 3:s lydelse)



Figur 1: Ett bevissystem för CTL.

Appendix

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

check_for_all(_, _, [], _, _).
check_for_all(T, L, [S|Tail], U, F) :-
    check(T, L, S, U, F),
    check_for_all(T, L, Tail, U, F).

check_for_some(T, L, [S|Tail], U, X) :-
    check(T, L, S, U, X);
    check_forsome(T, L, Tail, U, X).

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

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

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

% or1(F,G)
check(T, L, S, [], or(F,G)) :-
    check(T, L, S, [], F);
    check(T, L, S, [], G).

% AX
check(T, L, S, [], ax(F)) :-
    member([S, Neighbors], T),
```

```
    check_for_all(T, L, Neighbors, [], F).

% EX
check(T, L, S, [], ex(F)) :-
    member([S, Neighbors], T),
    check_for_some(T, L, Neighbors, [], F).

% AG1
check(_, _, S, U, ag(_)) :-
    member(S, U).

% AG2
check(T, L, S, U, ag(F)) :-
    \+member(S, U),
    check(T, L, S, [], F),
    member([S, Neighbors], T),
    check_for_all(T, L, Neighbors, [S|U], ag(F)).

% EG1
check(_, _, S, U, eg(_)):-
    member(S, U).

% EG2
check(T, L, S, U, eg(F)):-
    \+ member(S, U),
    check(T, L, S, [], F),
    member([S,Neighbors], T),
    check_for_some(T, L, Neighbors, [S|U], eg(F)).

% EF1
check(T, L, S, U, ef(F)) :-
    \+member(S, U),
    check(T, L, S, [], F).

% EF2
check(T, L, S, U, ef(F)) :-
    \+member(S, U),
    member([S,Neighbors], T),
    check_for_some(T, L, Neighbors, [S|U], ef(F)).
```

```
% AF1
check(T, L, S, U, af(F)) :-
    \+member(S, U),
    check(T, L, S, [], F).

% AF2
check(T, L, S, U, af(F)) :-
    \+member(S, U),
    member([S,Neighbors], T),
    check_for_all(T, L, Neighbors, [S|U], af(F)). `
```