25-07-16

EX1

Alphabet:

A(x), S(x), G(x), P(x), member(x,y), guest(x,y), recorded(x,y)

Axioms:

Forall x. G(x) implies A(x) \\ ISA

Forall x. P(x) implies A(x) \\ ISA

Forall x. G(x) implies not P(x) \\ disjoint

Forall x. A(x) implies G(x) or P(x) \\ complete

Forall x,y. member(x,y) implies P(x) and G(y) \\ typing

Forall y. G(y) implies 2<=#{x|member(x,y)} \\ multiplicity

Forall x,y. guest(x,y) implies P(x) and S(y) \\ typing

Forall x,y. recorded(x,y) implies A(x) and S(y) \\ typing

Forall y. S(y) implies 1<={x|recorded(x,y)}<=1 \\ multiplicity

EX2

1. We have an ISA in this UML diagram so we are going to follow this procedure to complete the instantiation.

Given UML diagram T

Iold = {}, Inew = I

While (Iold and Inew are different) do

For each (forall x. A(x) implies B(x) in T) do

For each (a in A^Inew) do

B^Inew = B^Inew union {a}

Similar for each subset constraints forall x,y. P(x,y) implies R(x,y)

I = Inew

Return I

I = (Obj^I, A^I, S^I, G^I, P^I, member^I, guest^I, recorded^I)

I0:

Obj^I = {Be,Rs,John, Paul,George, Ringo, Mick,Keith, I wanna be your man (original), I wanna be your man (cover)}

A^I = {}

S^I ={ I wanna be your man (original), I wanna be your man (cover)}

G^I = {Be,Rs}

P^I = { John, Paul,George, Ringo, Mick,Keith}

member^I = {(John,Be),(Paul,Be),(George, Be),(Ringo,Be),(Mick, Rs),(Keith,Rs)}

guest^I = {(John, I wanna be your man (original)), (Paul, I wanna be your man (cover)}

recorded^I = {(Rs, I wanna be your man (original)), (Be, I wanna be your man (cover)}

I1:

Each element of G is also element of A and also each element of P is also element of A

Obj^I = {Be,Rs,John, Paul,George, Ringo, Mick,Keith, I wanna be your man (original), I wanna be your man (cover)}

A^I = { Be,Rs , John, Paul,George, Ringo, Mick,Keith }

S^I ={ I wanna be your man (original), I wanna be your man (cover)}

G^I = {Be,Rs}

P^I = { John, Paul,George, Ringo, Mick,Keith}

member^I = {(John,Be),(Paul,Be),(George, Be),(Ringo,Be),(Mick, Rs),(Keith,Rs)}

guest^I = {(John, I wanna be your man (original)), (Paul, I wanna be your man (cover)}

recorded^I = {(Rs, I wanna be your man (original)), (Be, I wanna be your man (cover)}

I2:

Each element of G is also element of A and also each element of P is also element of A

Obj^I = {Be,Rs,John, Paul,George, Ringo, Mick,Keith, I wanna be your man (original), I wanna be your man (cover)}

A^I = { Be,Rs , John, Paul,George, Ringo, Mick,Keith }

S^I ={ I wanna be your man (original), I wanna be your man (cover)}

G^I = {Be,Rs}

P^I = { John, Paul,George, Ringo, Mick,Keith}

member^I = {(John,Be),(Paul,Be),(George, Be),(Ringo,Be),(Mick, Rs),(Keith,Rs)}

guest^I = {(John, I wanna be your man (original)), (Paul, I wanna be your man (cover)}

recorded^I = {(Rs, I wanna be your man (original)), (Be, I wanna be your man (cover)}

I2 = I1

The instantiation is complete

Now we are going to check if the instantiation is correct, so if I|=T. This means that we want to verify if all axioms in T are evaluated true in I.

Each element of G should have at least 2 member, so to instances of P in the association member, it is ok because Be has 4 and Rs has 2. Each instance of S is recorded by one and only one A. I wanna be your man (original) is recorded by Rs and I wanna be your man (cover) is recorded by Be, ok.

Each element of G should not be in P, and A contains elements that belong to G or to P, ok.

The instantiation is correct.

1. q(x)<- G(x) and Exists y,y’,y’’. P(y) and member(y,x) and P(y’) and member(y’,x) and P(y’’) and member(y’’,x) and y’ noteq y and y not eq y’’ and y’ noteq y’’

q(x):{Be}

1. q(x)<- P(x) and (forall y. S(y) and not recorded(x,y) implies guest(x,y)) or (Exists z. member(x,z) and forall k. S(k) and not recorded(z,k) implies guest(x,k)

q(x):{John}

1. q()<-not Exists y. S(y) and Exists z. G(z) and recorded(z,y) and Forall p. member(p,z) implies guest(p,y)

q():true

EX3

Model checking a closed mu calculus formula phi over transition system T = <S,Ra,Pi> (S is set of states, Ra is set of transitions, Pi mapping function from set a proposition to a subset of S) means that we want to verify if the initial state of T is in the extension of phi over T. With model checking we return a subset of S, where each element satisfies phi. To compute it we want to apply the labelling algorithm that consists in labelling states of T with predicates that are true in them. The extension of greatest fixpoint and least fixpoint are computed using Tarski-Knaster approximates theorem.

vX mu Y (b and <next> X) or (<next> Y)

We are going to find the greatest fixpoint (GFP) because of presence of vX

[X0] = {1,2,3,4,5}

[X1] = [muY (b and <next>X0) or <next>Y]

We are going to find the least fixpoint (LFP) because of presence of muY

[Y00]={}

[Y01] = [(b and <next>X0) or <next>Y00] = ([b] intersect PreE(next,[X0])) union PreE(next,[Y00]) = ({3,4} intersect {1,2,3,4,5}) union {} = {3,4}

[Y02] = [(b and <next>X0) or <next>Y01] = ([b] intersect PreE(next,[X0])) union PreE(next,[Y01]) = ({3,4} intersect {1,2,3,4,5}) union {1,2,3,4} = {1,2,3,4}

[Y03] = [(b and <next>X0) or <next>Y02] = ([b] intersect PreE(next,[X0])) union PreE(next,[Y02]) = ({3,4} intersect {1,2,3,4,5}) union {1,2,3,4} = {1,2,3,4}

[Y03] = [Y02] - - > found LFP

[X1] = {1,2,3,4}

[X2]= [muY (b and <next>X1) or <next>Y]

We are going to find the least fixpoint (LFP) because of presence of muY

[Y10]={}

[Y11] = [(b and <next>X1) or <next>Y10] = ([b] intersect PreE(next,[X1])) union PreE(next,[Y10]) = ({3,4} intersect {1,2,3,4}) union {} = {3,4}

[Y12] = [(b and <next>X1) or <next>Y11] = ([b] intersect PreE(next,[X1])) union PreE(next,[Y11]) = ({3,4} intersect {1,2,3,4}) union {1,2,3,4} = {1,2,3,4}

[Y13] = [(b and <next>X1) or <next>Y12] = ([b] intersect PreE(next,[X1])) union PreE(next,[Y12]) = ({3,4} intersect {1,2,3,4}) union {1,2,3,4} = {1,2,3,4}

[Y13] = [Y12] - - > found LFP

[X2] = {1,2,3,4}

[X2] = [X1] - - > found GFP

It is 1 in {1,2,3,4} ? Yes, so the formula is satisfied by this transition system

Now we are going to do model checking of CTL formula. Given a Kripke model (KM) = <S,I,R,AP,L> (S is set of states, I is set of initial state, R set of transitions, AP set of atomic propositions and L mapping function) and a CTL formula phi we want to check if KM,s|=phi where s is state of S. With model checking we return a subset of S that as I said before contains elements that satisfied phi. To compute this subset we are going to exploit the syntactic structure of CTL formula. In particular, we translate each sub formula of CTL in a mu calculus formula and then apply the labelling algorithm to find their extensions.

EG ( AX ( not a or AF b))

alpha = AF b = muX b or [next] X

beta = not a or alpha

gamma = AX beta

delta = EG gamma

[alpha] = [AF b] = [muX b or [next] X]

We are going to find the least fixpoint (LFP) because of presence of muX

[X0] = {}

[X1] = [b or [next]X0] = [b] union PreA(next,[X0]) = {3,4}

[X2] = [b or [next]X1] = [b] union PreA(next,[X1]) = {3,4} union {3,4} = {3,4}

[X2] = [X1] - - > found LFP

[alpha] = {3,4}

[beta] = [not a or alpha] = [not a] union [alpha] = {1,3} union {3,4} = {1,3,4}

[gamma] = [AX beta] = [[next] beta] = PreA(next,[beta]) = {3,4}

[delta] = [EG gamma] = [vX gamma and <next>X]

We are going to find the greatest fixpoint (GFP) because of presence of vX

[X0] = {1,2,3,4,5}

[X1] = [gamma and <next>X0] = [gamma] intersect PreE(next,[X0]) = {3,4} intersect {1,2,3,4,5} = {3,4}

[X2] = [gamma and <next>X1] = [gamma] intersect PreE(next,[X1]) = {3,4} intersect {1,2,3,4} = {3,4}

[X2] = [X1] - -> found GFP

[delta] = {3,4}

It is 1 in {3,4} ? No, so the CTL formula is not true in this transition system

EX4

Two states are bisimular if they have the same behaviour. This means that:

* Look undistinguishable
* Every action done in one state can also done in the other state

A binary relation R is a bisimularion iff (s,t) we show:

* s is final iff t is final
* For each action a
  + If s action a s’ then exists t’. t action a t’ and (s’,t’) in R
  + If t action a t’ then exists s’. s action a s’ and (s’,t’) in R

A state s0 of transition system is bisimular to a state t0 of transition system T If exists a bisimulation between s0 and t0.

The algorithm for bisimulation is the following:

1. R = SxT
2. R’ = R- {(s,t) such that not(s final in S noteq t is final in T)
3. While (R and R’ are different) do

R = R’

R’’ = R’-{(s,t) such that exists action a. s action a s’ and not exists t’. t action a t’ and (s’,t’) in R}{(s,t) such that exists action a. t action a t’ and not exists s’. s action a s’ and (s’,t’) in R}

Now we are going to solve the exercise.

1. Assuming that all states in S are equal to all states in T

R0 = {(s1,t1),(s1,t2),(s2,t1),(s2,t2),(s3,t1),(s3,t2)}

1. Remove pairs in which one state is final and the other one is not final

R1 = {(s1,t1),(s2,t2),(s3,t1)}

1. Remove iteratively pairs in which one state can do an action and the other state cannot copy it

R2 = {(s1,t1),(s2,t2)}

* t1 can do a and go to t2 but s3 cannot do a so I remove (s3,t1)

R3 = {(s1,t1)}

* t2 can do c and go to t1 and s2 can do c and go to s3 but (s3,t1) is not in R2 so I remove (s2,t2)

R4 = {}

* t1 can do a and go to t2 and s1 can do a and go to s2 but (s2,t2) is not in R3 so I remove (s1,t1)

R5 = {}

R5 = R4 - - > greatest fixpoint reached

S and T are not bisimular because (s1,t1) is not in R5

EX5

Check if Hoare triple {P} while g do S {Q} is correct cannot be done automatically but we need to apply inference rule.

Assuming to have a candidate invariant I we show:

* P implies I
* {I and g} S {I} = {I and g} implies wp(S,I}
* {I and not g} implies Q

If candidate I respect these operations is called invariant

If we have the candidate these operations are computed automatically, otherwise it is difficult to mange with them. The problem of checking if Hoare triple is correct is undecidable because there is no sound or complete techniques to do this or to find all possible invariants. The only thing we can do is to check if the candidate is invariant or not.

Candidate: i<=10

{i=0} while(i<10} do i:=i+1 {i=10}

* P implies I

{i=10} implies {i<=10}

Ok because if i=0 is always less or equal than 10

* {I and not g} implies {Q}

{i<=10 and i>=10} implies {i=10}

{i=10} implies {i=10}

Ok

* We are going to compute wp(S,I)

{i<=9} is wp

i=i+1

{i<=10}

{I and g} implies wp(S,I)

{i<=10 and i<10} implies {i<=9}

{i<10} implies {i<=9}

The condition is satisfied because if i<10 it would be always less or equal than 9. So {i<=10} is invariant